Demystifying Transformer: A Case Study on Graph-Based Interpretability and Attention Layer Dynamics

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Master of Technology
in
Computer Science & Engineering

by

Purvaja Kale (MT23MCS005)

under the guidance of

Dr. Ashish Tiwari

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

Demystifying Transformer: A Case Study on Graph-Based Interpretability and Attention Layer Dynamics

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfillment of the requirements for the award of
the degree

Master of Technology
in
Computer Science & Engineering

by

Purvaja Kale (MT23MCS005)

under the guidance of

Dr. Ashish Tiwari

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology
Nagpur 440 010 (India)

JUNE 2025

©Visvesvaraya National Institute of Technology (VNIT) 2025

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology, Nagpur

Declaration

I, **Purvaja Kale**, hereby declare that this dissertation titled **" Demystifying Transformer: A Case Study on Graph-Based Interpretability and Attention Layer Dynamics "** is carried out by me in the Department of Computer Science & Engineering of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier whole or in part for the award of any degree at this or any other Institution.

Purvaja Kale
Date: 30th May, 2025

Certificate

This to certify that the dissertation titled **" Demystifying Transformer: A Case Study on Graph-Based Interpretability and Attention Layer Dynamics "** is submitted by **Purvaja Kale** in partial fulfillment of the requirements for the award of the degree of **Master of Technology in Computer Science & Engineering**, VNIT Nagpur. The work is comprehensive, complete, and fit for final evaluation.

Dr. Ashish Tiwari
Assistant Professor
Department of Computer Science & Engineering
VNIT, Nagpur

Dr. M.P. Kurhekar
Head, Department of Computer Science & Engineering
VNIT, Nagpur

Department of Computer Science & Engineering
Visvesvaraya National Institute of Technology, Nagpur

Non-Plagiarism Certificate

Certified that M. Tech Dissertation titled " **Demystifying Transformer : A Case Study on Graph-Based Interpretability and Attention Layer Dynamics** " submitted by **Purvaja Kale**, Enrollment No. **MT23MCS005**, ID No. **30125** has been checked for plagiarism using TURNITIN software and the overlap is found within prescribed limits. A summary of the report is given below.

Purvaja_MTech_Thesis.pdf

ORIGINALITY REPORT

14%	11%	10%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

Purvaja Kale
Enrollment No: MT23MCS005

Dr. Ashish Tiwari
Assistant Professor
Department of Computer Science & Engineering
VNIT, Nagpur

ACKNOWLEDGEMENT

I would like to take this opportunity to express my profound gratitude to all those who have contributed to the successful completion of my M.Tech dissertation.

First and foremost, I am deeply thankful to my supervisor, Dr. Ashish Tiwari, for his constant guidance, support, and encouragement. His wealth of knowledge, thoughtful insights, and patient mentorship have played a pivotal role in shaping the direction and quality of this work. I feel truly privileged to have worked under his supervision.

My sincere thanks are also extended to the esteemed faculty of the Department of Computer Science & Engineering, Visvesvaraya National Institute of Technology, Nagpur, for their continuous support and for fostering an environment that encourages academic rigor and intellectual curiosity. Their valuable inputs, whether through coursework or discussions, have enriched my learning and significantly contributed to my research.

I am also immensely grateful to my family and friends, whose unwavering support, understanding, and motivation have been my strength throughout this journey. Their belief in me has been a source of inspiration during moments of challenge and growth.

To each and every person who has been a part of this academic journey—thank you.

Thank you all.

Purvaja Kale

ABSTRACT

This paper presents a detailed case study aimed at unraveling the inner workings of the Transformer architecture by exploring how individual layers and attention heads collaboratively contribute to accurate prediction. A key component of their success lies in the attention mechanism, which plays a pivotal role in capturing contextual relationships across tokens and preserving learned knowledge across multiple layers. Understanding this mechanism is crucial, as it determines how information flows and influences decisions during inference.

The primary objective of this research is to systematically study how each layer in a transformer model processes and refines information, ultimately leading to prediction. While Transformers have been widely adopted in practical applications, their internal decision-making processes still remain opaque—often referred to as a "black-box" problem in deep learning. By peeling back the layers of abstraction, we aim to provide a more transparent view of how the model builds understanding and makes predictions.

To support our investigation, we leverage a variety of visualization and interpretability tools. These include heatmaps to trace attention distributions, graph-based representations to understand inter-token connectivity, and clustering techniques to observe pattern formations across tokens and layers. Our experiments are conducted on machine translation tasks, which serve as an ideal testbed due to their dependence on contextual understanding and syntactic alignment. Through empirical analysis and visual exploration, we aim to validate the hypothesis that attention-based architectures offer interpretable pathways to understanding model behavior, particularly in how semantic meaning is encoded and transferred across layers.

LIST OF FIGURES

3.1 Example of English-Hindi sentence pairs from the dataset. 6
3.2 Implementation Flowchart 8

4.1 Tokens plotted on Graph 10
4.2 Similarity of tokens 11
4.3 Position of tokens based on Old-coordinates 12
4.4 Position of tokens based on New-coordinates 12
4.5 context pull 1 13
4.6 Context pull 2 13
4.7 Context pull 3 13
4.8 Transformer 15
4.9 Visualizing sinusoidal positional encoding variations 18
4.10 Summation Step 19
4.11 Self-attention 20
4.12 Linear Transformation matrix derived from Q and K 21
4.13 Visual representation of transformation 22
4.14 Transformation by V 22
4.15 Multi-Head Attention 24
4.16 Multiple single head attentions 25
4.17 Concatenation 25
4.18 Multi-head concatenation 26
4.19 Multi-head Attention output 26
4.20 FFN Layers 28
4.21 Feed Forward Neural Network 29
4.22 Decoder 30

5.1 Heat Map example 34
5.2 Heat Map Visualisation 36
5.3 Graphical Visualisation at Layer 5 Head 0 37
5.4 Cluster Analysis Example 39
5.5 Clusters 39

5.6 Betweeness Centrality 40
5.7 Important Nodes in Encoder and Decoder 42
5.8 BLEU Score Analysis 44

LIST OF TABLES

4.1 Token Coordinates in Technology and Fruitness Space 9
4.2 Similarity Matrix 11
4.3 Tokenized words 15
4.4 Randomly Initialized Word Embeddings 17
4.5 Sample positional encoding values for the sentence ”I bought an apple” . 18
4.6 Word Embedding Matrix 19
4.7 Positional Embedding Matrix 19
4.8 WP Matrix (Word + Positional Embedding) 19
4.9 Comparison of Q, K, and V matrices 23
4.10 Comparison of Multi-head Attention Types in Transformer Architecture . 27

NOMENCLATURE

Symbols

α	Learning rate used in gradient descent
d_k	Dimension of query and key vectors
d_v	Dimension of value vectors
d_{ff}	Dimension of the feed-forward network
d_{model}	Input embedding dimension (typically the same across all layers and heads)
H	Number of attention heads
K	Key matrix
N	Number of layers in the Transformer model
N_x	Number of encoder/decoder layers
P	Positional encoding matrix
Q	Query matrix
t	Context window size (sequence length)
V	Value matrix
W	Word embedding matrix
W_K	Weight matrix for Key
W_O	Output projection weight matrix
W_Q	Weight matrix for Query
W_V	Weight matrix for Value

WP	Combined embedding matrix, $W + P$
WP^T	Transpose of combined embedding matrix
Norm	Layer normalization

Acronyms

BERT	Bidirectional Encoder Representations from Transformers – a masked language model
BLEU	Bilingual Evaluation Understudy – translation quality metric
GPT	Generative Pretrained Transformer – an autoregressive language model
NLP	Natural Language Processing – a field of artificial intelligence focused on the interaction between computers and human language
T5	Text-to-Text Transfer Transformer – a unified framework that converts all NLP tasks into a text-to-text format

INDEX

ABSTRACT i

LIST OF FIGURES ii

LIST OF TABLES iv

NOMENCLATURE v

INDEX vii

1 INTRODUCTION 1

1.1 Overview 1

1.2 Problem Statement 1

1.3 Objectives 2

1.4 Thesis Outline 2

2 LITERATURE SURVEY 4

3 METHODOLOGY 6

3.1 Dataset 6

3.2 Model Used 6

3.2.1 Tokenizer 6

3.2.2 Model Architecture 7

3.2.3 Training Details 7

3.3 Implementation Workflow 7

4 UNDERSTANDING TRANSFORMERS 9

4.1 Context Pull 9

4.2 Introduction to Transformers 13

4.2.1 Tokenization / Encoding 14

4.2.2 Embedding 16

4.2.3 Positional Encoding 17

4.2.4 Attention Mechanism 19
4.2.5 Adding and Normalization 27
4.2.6 Feed Forward Neural Network 28
4.2.7 Decoder 29
4.3 Transformer Dimension Flow Summary 32

5 ANALYSIS AND VISUALIZATION OF ATTENTION 34
5.1 Heat Maps 34
5.2 Graphs 35
5.3 Cluster 38
5.3.1 Detecting communities (clusters): 38
5.3.2 Betweenness Centrality: 39
5.3.3 Top-k Important Nodes: 41
5.4 BLEU Score Analysis 41

6 CONCLUSION AND FUTURE SCOPE 46
6.1 Conclusion 46
6.2 Future Scope 46

Chapter 1

INTRODUCTION

1.1 Overview

In recent years, Natural Language Processing (NLP) has seen a transformative shift with the advent of deep learning techniques. Traditional models such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were previously at the forefront of sequence modeling. However, these architectures suffer from limitations like vanishing gradients, limited memory for long-range dependencies, and difficulty in parallelization.

The introduction of the Transformer architecture by [1] in 2017 marked a paradigm shift in the field of NLP and sequence modeling. Unlike previous models, Transformers rely entirely on the attention mechanism, eliminating recurrence altogether. This shift enabled better performance, scalability, and parallelization, making Transformers the foundation of many state-of-the-art models like BERT, GPT, T5, and more.

Despite their success, the internal operations of Transformers, particularly the behavior of attention mechanisms across multiple layers and heads, remain largely opaque. Understanding how these models learn and make decisions is essential to advance interpretability and trust in AI systems.

1.2 Problem Statement

While Transformer models have achieved remarkable performance across a wide range of NLP tasks, their decision-making processes remain largely a black box. The attention mechanism, although central to these models, is difficult to interpret directly without proper tools or visualization techniques.

There is a lack of comprehensive understanding of how different attention heads and layers contribute to learning specific linguistic or semantic features. Without this understanding, it becomes challenging to interpret model behavior, diagnose errors, or improve architectures systematically.

This research aims to bridge this gap by developing interpretability tools and visualization techniques that can expose the learning dynamics of attention mechanisms within Transformer architectures.

1.3 Objectives

The primary goal of this research is to deepen the understanding of transformer-based models by focusing on the attention mechanism. The key objectives are:

1. **Explain Transformer components:** Provide an overview of the architecture, including encoder-decoder structure, multi-head attention, positional encoding, feed-forward layers, and normalization.

2. **Investigate the role of attention:** Analyze how self-attention and cross-attention help the model focus on relevant parts of input sequences and capture contextual relationships.

3. **Visualize learning across layers and heads:** Use techniques like attention heatmaps and token interaction graphs to illustrate how information is processed and aggregated during training and inference.

4. **Analyze attention scores with evaluation metrics:** Correlate attention patterns with metrics such as BLEU scores to interpret model decisions and explore the explainability of transformers.

These insights aim to identify redundant attention heads, suggest architectural improvements, and enhance trust by explaining model behavior. Additionally, this interpretability supports educational efforts by clarifying transformer internals for learners and researchers.

1.4 Thesis Outline

Chapter 1: Introduction

This chapter provides an overview of the research topic, outlines the objectives of the thesis, and presents the structure of the report. It sets the stage for understanding transformer-based models and highlights the motivation for focusing on attention analysis and interpretability.

Chapter 2: Literature Survey

This chapter reviews significant research contributions related to transformer models, attention mechanisms, and visualization techniques. It lays the foundation for the problem space and identifies key gaps that this thesis aims to address.

Chapter 3: Methodology

This chapter discusses the research methodology, including the specific transformer model used, dataset selection, and the overall experimental workflow.

Chapter 4: Understanding Transformers

This chapter offers a deep dive into the architecture of transformer networks. It explains the fundamental components such as multi-head attention, positional encoding, and context pulling with the help of code snippets and diagrams. The chapter also covers how layers are assembled and how information is propagated through them.

Chapter 5: Analyzing and Visualization of Attention

This chapter presents the analysis and interpretation of attention mechanisms using various visualization tools. It includes heatmaps, attention graphs, and cluster analysis to uncover how the model learns. Additionally, epoch-wise numerical trends and BLEU scores are reported to evaluate translation quality and learning behavior.

Chapter 6: Conclusion and Future Scope

This chapter summarizes the key findings of the thesis and reflects on the effectiveness of visualization techniques in understanding transformer attention. It also proposes potential areas for future research in improving interpretability and performance analysis of large language models.

Chapter 2

LITERATURE SURVEY

The attention mechanism was first introduced by *"Bahdanau et el (2015)"* [2] in *"Neural Machine Translation by Jointly Learning to Align and Translate"* to address the performance bottlenecks of conventional encoder-decoder architectures in sequence-to-sequence tasks. Their work significantly enhanced machine translation by allowing the decoder to focus on specific parts of the input sequence dynamically. Building on this concept, *"Vaswani et el (2017)"* [1] proposed the Transformer architecture in *"Attention is all you Need"*, which relies entirely on self-attention mechanisms, bypassing the limitations of recurrence and convolutions. This paradigm shift enabled highly parallelizable and scalable models, paving the way for the rise of Large Language Models (LLMs).

However, while the Transformer architecture demonstrated remarkable empirical success, the original paper lacked a detailed justification of the internal architecture and did not thoroughly explain the rationale behind the design of its components. This has led to multiple interpretations of how the architecture works and limited understanding of how individual components contribute to predictions. As a result, optimization remains a challenge, especially due to the high computational and memory requirements of the model.

The paper titled *"Attention is not Explanation"* [3] delved into the question of whether attention mechanisms provide meaningful explanations for model predictions. It critically examined whether attention weights can be reliably interpreted as explanations. However, their evaluations were limited in scope, focusing on tasks with unstructured output and not considering sequence-to-sequence tasks. Their analysis was particularly centered on RNNs and commonly used attention techniques.

Review papers such as *"A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges"* [4] and *"A Comprehensive Overview of Large Language Models"* [5] have provided broad insights into the architecture, classification, and evolution of LLMs. These surveys also categorize LLMs based on their suitability for various applications, emphasizing that a deeper understanding of attention mechanisms is crucial for effectively leveraging LLMs based on application-specific requirements. *"A*

Survey of the State of Explainable AI for Natural Language Processing" [6] identified the growing demand for explainability and proposed a classification scheme for various explainability techniques. *"Explainability for Large Language Models: A Survey"* [7] introduced two main paradigms to approach explainability, though it did not offer strong logical reasoning to explain why LLMs behave a certain way.

The paper *"Faithfulness vs. Plausibility: On the (Un)Reliability of Explanations from Large Language Models"* [8] introduced the distinction between plausible and faithful explanations in the self-explanation behavior of LLMs. It also described three key self-explanation methods and highlighted the reliability issues of LLM-generated explanations.Works like *"Chain-of-Thought Prompting Elicits Reasoning in Large Language Models"* [9] explored how prompt engineering can help elicit reasoning paths in LLMs, contributing to the broader narrative of interpretability.*"On the Interpretability of Attention Networks"* [10] suggested quantifiable measures for interpretability in the context of image captioning, although this was not directly extended to machine translation models.

Papers such as *"Rethinking Interpretability in the Era of Large Language Models"* [11] and *"XAI meets LLMs: A Survey of the Relation between Explainable AI and Large Language Models"* [12] emphasize the importance of dataset explanation and interactive explanations as emerging research priorities. These works underline the pressing need for interpretability in LLMs to address issues such as user trust, transparency, potential misuse, critical thinking, ethical and privacy concerns, hallucinations, and inaccuracies.

In conclusion, the reviewed literature highlights a critical and timely need to understand and validate how attention mechanisms operate, and how the architecture of Transformers facilitates this process. This sets the foundation for our research which aims to interpret and visualize attention in Transformer models using meaningful metrics and visual tools.

Chapter 3

METHODOLOGY

3.1 Dataset

The English training sentences are sourced from the Kaggle dataset titled *English to Marathi Machine Translation Dataset* [13]. The dataset originally contains 3,627,480 sentences. We converted these English sentences to Hindi using the Bing Translation API for our target language. The sentences are saved in the form of id, translation in English and then in Hindi, as shown in Fig:3.1

- Total sentences translated: 50,584


```
{
    "id": 10,
    "translation": {
        "en": "He says India is special to him.",
        "hi": "उनका कहना है कि भारत उनके लिए खास है।"
    }
}
,
{
    "id": 11,
    "translation": {
        "en": "For me, the priority is to put every penny received from the taxes paid by the honest and
        hard working tax payers to right use.",
        "hi": "मेरे लिए प्राथमिकता ईमानदार और मेहनती करदाताओं द्वारा दिए गए करों से प्राप्त हर पैसे को
        सही उपयोग में लाने की है।"
    }
}
```


Figure 3.1: Example of English-Hindi sentence pairs from the dataset.

3.2 Model Used

3.2.1 Tokenizer

We use the `WordLevel()` tokenizer from Hugging Face to perform word-level tokenization on all sentences in the dataset.

3.2.2 Model Architecture

The transformer model used follows the original architecture as described in the seminal paper vaswani, with the following hyperparameters:

- Input embedding dimension, $d_{model} = 512$
- Dimensions of query, key, value weights, $d_k = d_v = d_{model}/h = 64$
- Number of attention heads, $h = 8$
- Number of encoder/decoder layers, $N_x = 6$
- Dimension of feed-forward layer, $d_{ff} = 2048$
- Context window size (sequence length), $seq_{len} = 200$ (depends on max input length)

3.2.3 Training Details

- Number of epochs: 30
- Batch size: 16
- Learning rate: 10^{-4}
- Steps per epoch: 2,846
- Total training steps: $30 \times 2,846 = 85,380$
- Approximate training time: 20 hours on an RTX 4050 GPU (6 GB VRAM)

3.3 Implementation Workflow

This project, as described in flowchart:3.2, begins with understanding the **Transformer architecture**, focusing on components like **attention mechanisms** and **context pull**, which explain how tokens influence each other during learning.

We then explore various **visualization and analysis techniques**:

- **Heat Maps** to observe token-wise attention patterns,
- **Graph Analysis** to model token relationships,
- **Cluster Analysis** to group related tokens, and
- **BLEU Score** to evaluate model output quality.

Advanced graph-based methods like **cluster detection**, **betweenness centrality**, and identifying **Top-K important nodes** help pinpoint which tokens are most influential. We also analyze **when the model begins meaningful learning** across layers.

The combination of structural and performance-based analyses provides deep insight into how Transformers learn and make decisions.

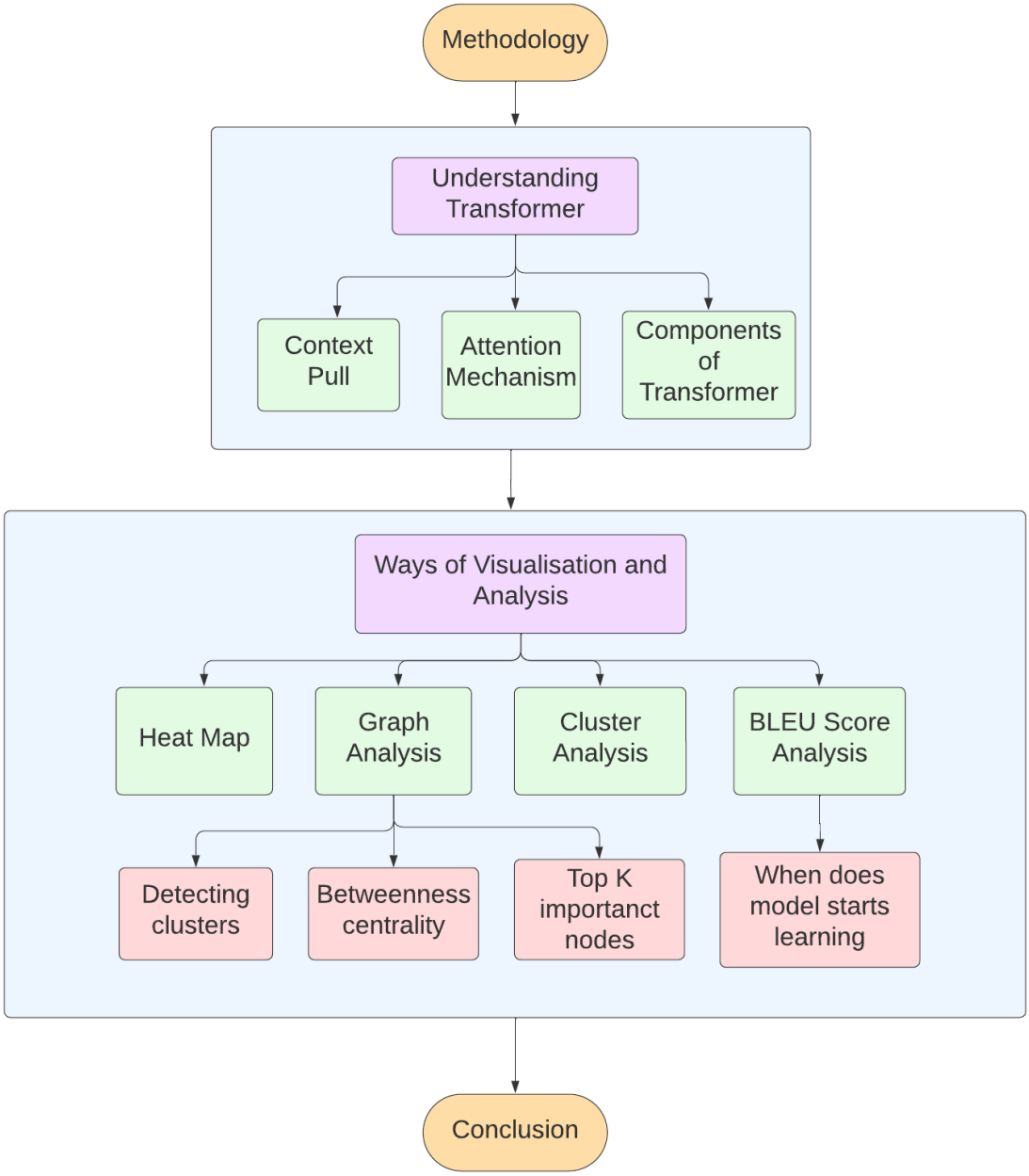


Figure 3.2: Implementation Flowchart

Chapter 4

UNDERSTANDING TRANSFORMERS

Here, we will discuss and explore the **context pull**, **attention mechanism**, and the fundamental **components of the Transformer** architecture, to understand their roles in the model's learning process.

4.1 Context Pull

Context pull in the framework of attention mechanisms and Transformer models describes how each token dynamically "pulls" relevant information from other tokens in the input sequence. This process allows the model to weigh the importance of different tokens contextually, enabling it to capture long-range dependencies and relationships effectively. The attention mechanism operationalizes this by computing attention scores that determine how strongly each token influences others during encoding and decoding.

Original paper follows attention mechanism concept given by Bahdanau et el [2]. Here the attention mechanism concept can be understood as pull of tokens towards each other with a certain attention score between them, which can be derived using similarity score.

Consider the following sentences in our datatset, where token "Apple" serves different purposes in both the sentences:
"An apple and an orange."
"An apple phone".

Let each word be a token represented as a vector in a 2-dimensional imaginary coordinate system with axes representing *Technology* and *Fruitness*, represented in Table:4.1.

Table 4.1: Token Coordinates in Technology and Fruitness Space

Token	Technology	Fruitness
Apple	2	2
Orange	0	3
Phone	4	0

Let's focus on noun words rather than common words like prepositions, articles etc (eg: 'a', 'the', 'and'). After plotting the tokens on coordinate axes, we can observe the distances between token in Fig:4.1

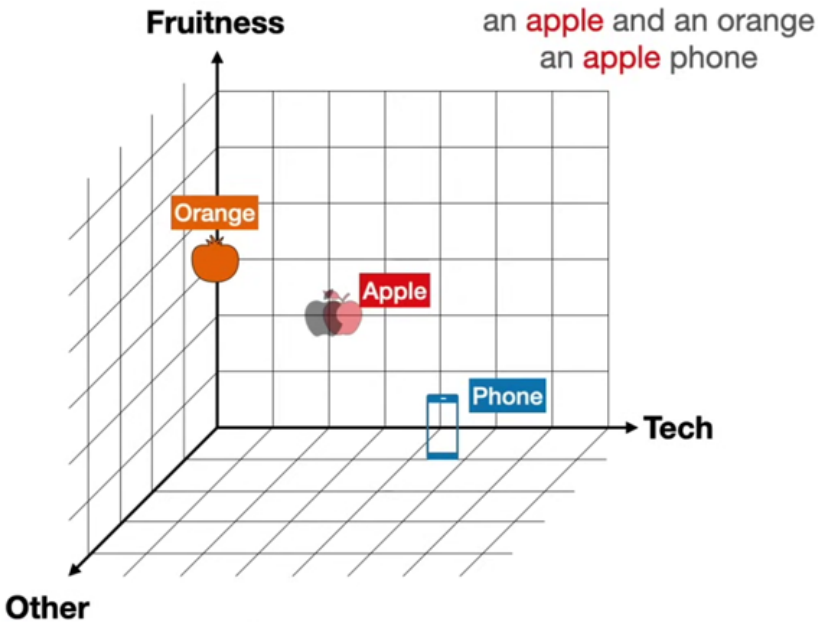


Figure 4.1: Tokens plotted on Graph

To compute the similarity between tokens (e.g., for attention computation), we use the dot product of their vectors. This allows us to express how much one word attends to another, based on their alignment in this conceptual space. We can visually interpret how similarity score is calculated of any two tokens is show in Fig:4.2. Where a similarity is nothing but product of one vector with transpose of another.

The equation to calculate measure of similarity is given as:

$$\text{Similarity}(A, B) = \frac{A \cdot B}{\sqrt{2}}$$

Where A and B are vectors, and divided by $\sqrt{2}$, to scale the similarity measure, where 2 signifies the dimension of a single vector.

Now, after calculating similarity measure between each token with every other token will produce the table like Table:4.2. This similarity values how each token is similar to other, higher the score higher the similarity.

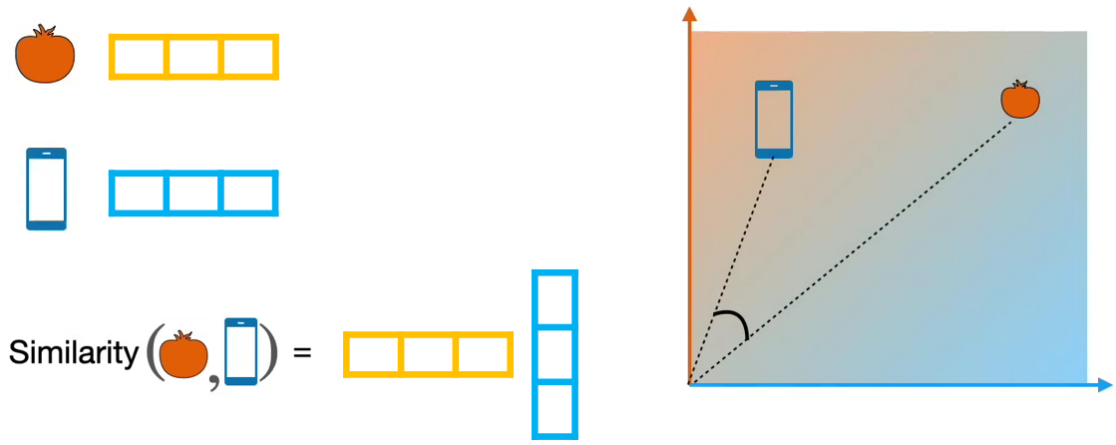


Figure 4.2: Similarity of tokens

Table 4.2: Similarity Matrix

From \ To	Apple	Orange	Phone
Apple	5.66	4.24	5.66
Orange	4.24	6.36	0.00
Phone	5.66	0.00	11.31

Referring to the above calculated similarity values from Table:4.2, we can express each token with respect to all the other tokens present in the token in the following manner:

$$\text{Apple} \rightarrow 5.66 \cdot \text{Apple} + 4.24 \cdot \text{Orange}$$

$$\text{Apple} \rightarrow 5.66 \cdot \text{Apple} + 5.66 \cdot \text{Phone}$$

This denotes that *Apple* is similar to 6.36 times itself and 4.24 times *Orange*. Since a token is always most similar to itself, it receives the highest similarity value, which is 6.36. While normalizing, there might be some instances where similarity values become negative, leading to a zero denominator. To avoid this, we use the Softmax function:

$$x \rightarrow e^x$$

For example, for *Apple*:

$$\text{Apple} \rightarrow \frac{e^{4.24} \cdot \text{Orange} + e^{5.66} \cdot \text{Apple}}{e^{4.24} + e^{5.66}} = 0.195 \cdot \text{Orange} + 0.805 \cdot \text{Apple}$$

$$\text{Apple} \rightarrow \frac{e^{5.66} \cdot \text{Phone} + e^{5.66} \cdot \text{Apple}}{e^{5.66} + e^{5.66}} = 0.5 \cdot \text{Phone} + 0.5 \cdot \text{Apple}$$

Here, we calculate the coordinates for *Apple* based on the influence of words that occur in the same sentence as the word. After applying softmax, we can replace the original coordinate values with the related token coordinates to get new coordinates for the token Apple for two sentences:

$$\text{Apple} = 0.195 \times \text{Orange} + 0.805 \times \text{Apple} = [0, 0.585] + [1.61, 1.61] = [1.61, 2.195]$$

$$\text{Apple} = 0.5 \times \text{Phone} + 0.5 \times \text{Apple} = [2, 0] + [1, 1] = [3, 1]$$

After plotting old coordinates of old tokens we get to see graph like fig4.3 and after performing above process of calculating new coordinates for tokens repeatedly we get graphs like Fig:4.4 :

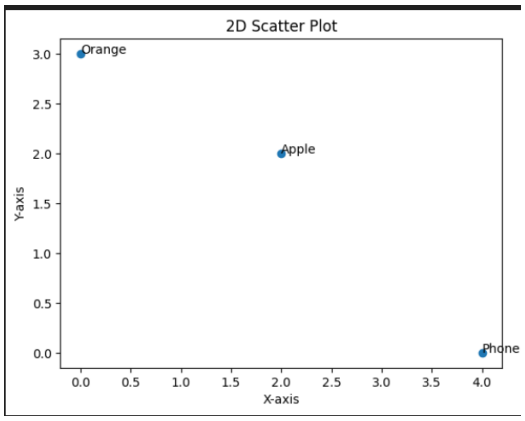


Figure 4.3: Position of tokens based on Old-coordinates

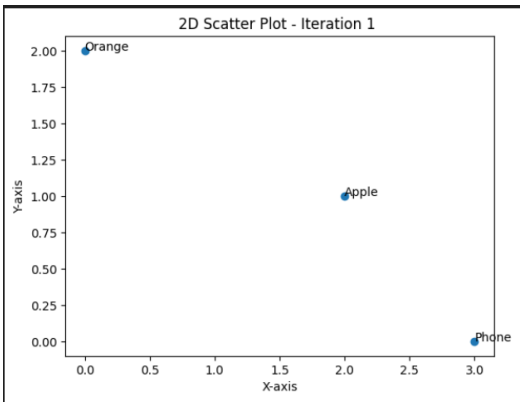


Figure 4.4: Position of tokens based on New-coordinates

The new coordinates of apple are closer to the coordinates of either orange or apple depending on which sentence apple appears and hence we have a better version of the token apple.This attention step is applied many times in a transformer as it trains.

We can visually represent context pull as Fig:4.5, 4.6, 4.7 with the help of some dummy tokens, it describes how tokens move across plane after repeatevtive changes in their coordinates:

Hence the words are going to end up much closer to what is dictating the context in that piece of text. This is what attention is!

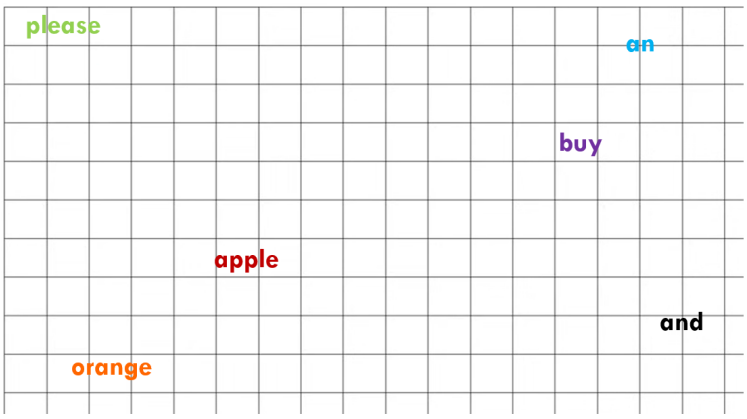


Figure 4.5: context pull 1

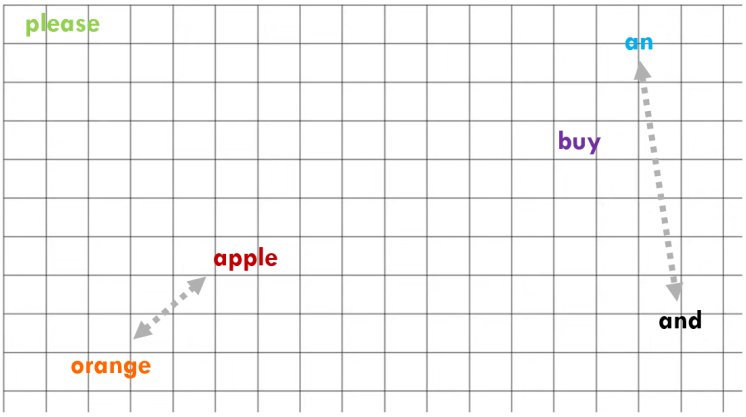


Figure 4.6: Context pull 2

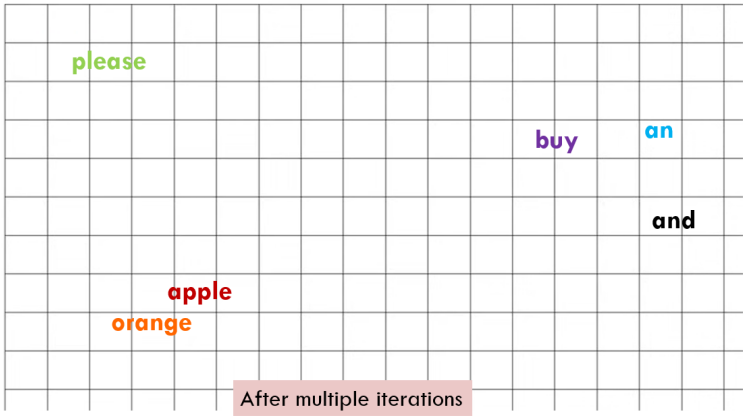


Figure 4.7: Context pull 3

4.2 Introduction to Transformers

Transformers played a vital role in overcoming Recurrent Neural Network (RNN) limitations such as tracking long-range dependencies, gradient vanishing and explosion, the number of training steps, and the requirement for sequential computation. The core idea

behind the transformer is the use of self-attention mechanisms that process each word in relation to all other words in a sentence. This is contrary to traditional methods that process words in sequential order.

Let us now understand the following components of a transformer in detail as given in fig:4.8 from [1]:

1. Tokenization / Encoding
2. Embeddings
3. Positional Encoding
4. Attention Mechanism
   (a) Self-attention
   (b) Cross-attention
   (c) Multi-head attention
   (d) Masked multi-head attention
5. Adding and Normalization
6. Feed Forward Neural Network
7. Decoder

The Fig:4.8 contains the architecture of transofomer, containing encoder and decoder blocks. The figure also consists of Multi-head attention describing scaled dot-product attention with each step how it is calculated. We will be discussing each component given in the architecture with the help of an example.

4.2.1 Tokenization / Encoding

Transformers take inputs in the form of tokens. Tokenization consists of breaking down text into tokens, which can be words or letters, depending on the method used.

Tokens help the model understand the actual elements in the input and assist in associating relationships between them. While many methods for tokenization exist, we will discuss how word-based tokenization is achieved.

Consider the following lowercase dataset:

```
"i bought an apple phone.  the basket contains an apple
and an orange."
```

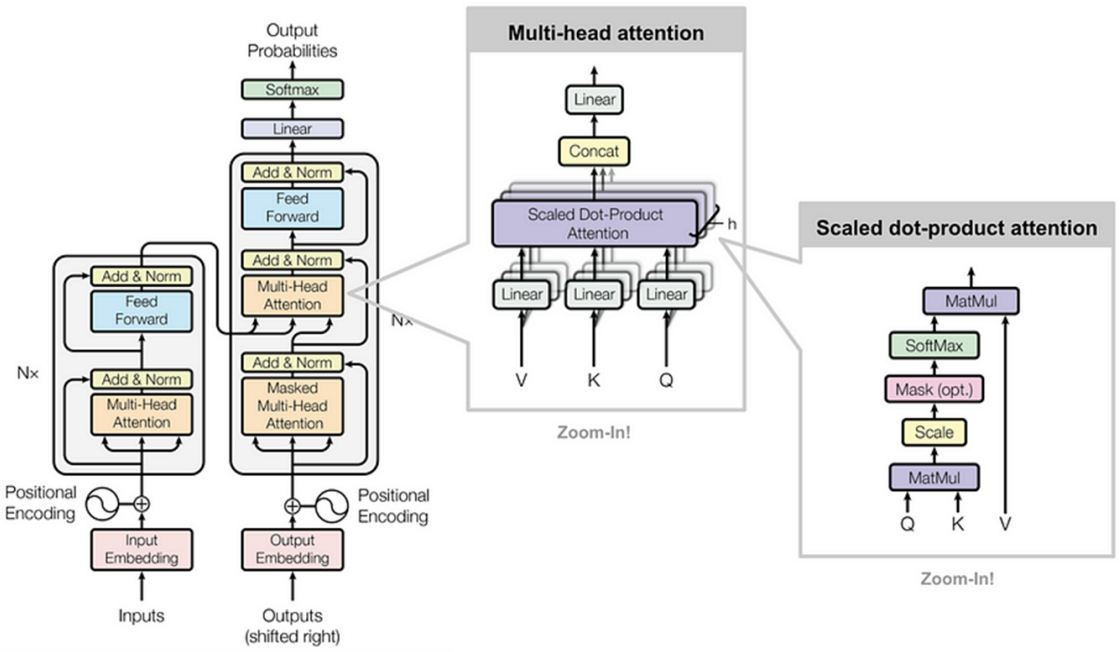


Figure 4.8: Transformer

Encoding unique words and serializing them (word-based encoding):

Vocabulary size:

$$\text{Vocab Size} = |\text{Set of Unique Words}| = 10$$

Table:4.3 represents each token of our dataset with respective token IDs. These token IDs are initialised randomly.

Table 4.3: Tokenized words

Word	Token ID
apple	0
phone	1
i	2
an	3
the	4
basket	5
bought	6
contains	7
and	8
orange	9

Types of Tokenization Methods

- **Character-based Tokenization:**

- *Description:* Splits text into individual characters. This method handles OOV words and typos better but increases sequence length significantly.
- *Example:*
  Input: ``apple''
  Output: [``a'', ``p'', ``p'', ``l'', ``e'']

- **Subword-based Tokenization:**
  - *Description:* Combines the advantages of both word and character tokenization by breaking words into frequently occurring subword units. This method is used in BERT, GPT, and other transformer models.
  - *Example 1 (Word: "unaffordable"):*
    Output: [``un'', ``afford'', ``able'']
  - *Example 2 (Sentence):*
    Input: ``i bought an apple phone''
    Output (BPE): [``i'', `` bought'', `` an'', `` apple'', `` phone'']

Each of these tokenization strategies has its own strengths and trade-offs. Word-based tokenization is straightforward, character-based is flexible, and subword-based is efficient and generalizes well across tasks. GPT-3[1] uses a large token vocabulary.

4.2.2 Embedding

Embeddings are mathematical depictions of relationships between tokens in a multi-dimensional space. These help the model understand complex relationships and encode them into a single vector. Word embedding vector values are initially filled with random values between 0 and 1. As training progresses, these values are updated as the Transformer learns the meanings and relationships among words.

Example: Using a $d_k = 4$ dimensional embedding vector, the word embeddings for the sentence "*i bought an apple*" are randomly initialized as given in Table:4.4:

Word Embedding Matrix: Although word embedding vector values are initially between -1 and 1 (or sometimes between 0 and 1), they are randomly initialized. These vectors are then fine-tuned during training to capture semantic relationships. The Transformer architecture proposed in the original paper uses a 512-dimensional embedding vector for each input word:

$$\text{Embedding Matrix} \in \mathbb{R}^{d_{\text{model}} \times n}, \quad \text{where } d_{\text{model}} = 512, \text{n = no. of tokens}$$

[1] GPT-3 has a vocabulary size of 50,257 tokens.

Table 4.4: Randomly Initialized Word Embeddings

Token	*i:2*	*bought:6*	*an:3*	*apple:0*
Row 1	0.23	0.34	0.33	0.61
Row 2	0.67	0.79	0.22	0.58
Row 3	0.47	0.92	0.42	0.51
Row 4	0.21	0.17	0.34	0.76

4.2.3 Positional Encoding

In languages, the **order of the words** and their **position** in a sentence really matters. The meaning of the entire sentence can change if the words are re-ordered.

Example: "The teacher helped the student." is different from "The student helped the teacher." although both sentences have the same words.

When implementing NLP solutions, **Recurrent Neural Networks (RNNs)** have an inbuilt mechanism that deals with the order of sequences. However, the **Transformer model** does not use recurrence or convolution and treats each data point as independent of the other. Hence, **positional information** is added to the model explicitly to retain the information regarding the order of words in a sentence. *Positional encoding* is the scheme through which the knowledge of the order of objects in a sequence is maintained.

Positional Embedding for any token is given as:

- For even positions in a sentence:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

- For odd positions in a sentence:

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

where d_{model} is the dimension of the embedding vector.

Frequency Term $\left(10000^{2i/d_{\text{model}}}\right)$:

- Controls the *wavelength* of sinusoidal functions.
- Forms a *geometric progression* of wavelengths from 2π (short wavelength) to $2\pi \times 10000$ (long wavelength).
- Lower dimensions (small i) capture *high-frequency* (local) positional information.
- Higher dimensions (large i) capture *low-frequency* (global) positional information.

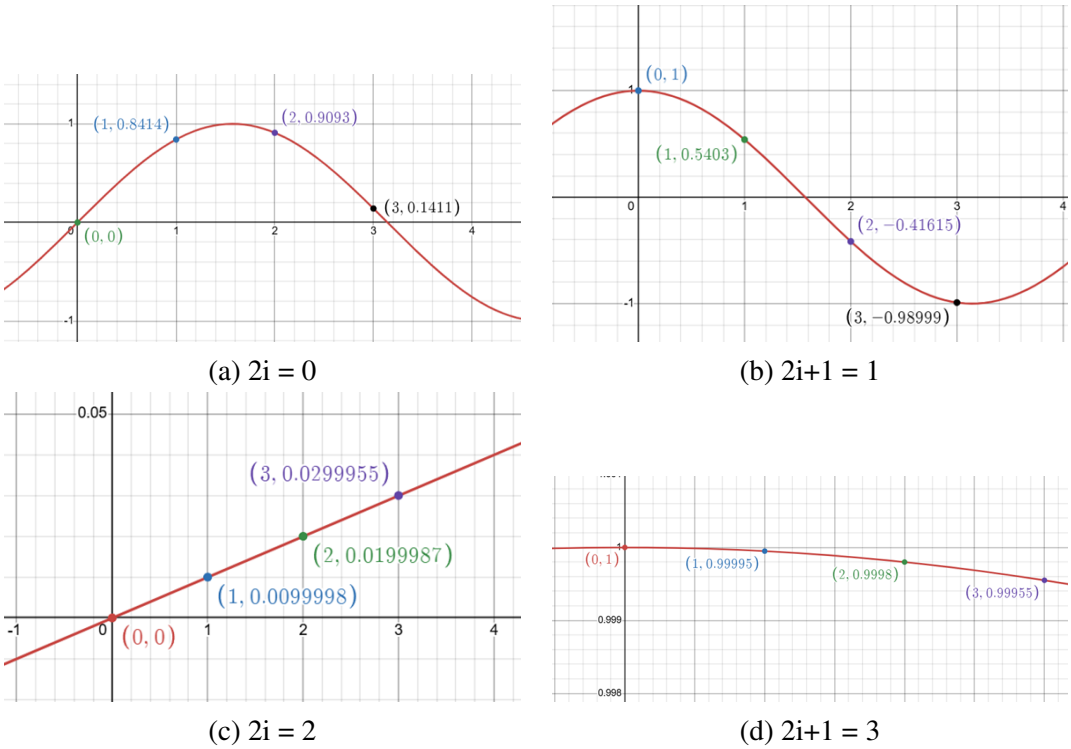
This geometric progression of wavelengths ensures the model handles both local and global dependencies. Using both **sine and cosine** functions ensures that each position gets a unique encoding. Alternating sine and cosine avoids symmetry issues and provides complementary information, as sine and cosine are phase-shifted versions of each other, enriching the positional representation.

Continuing the example: `"i bought an apple"`. We get positional embedding as Table:4.5:

Table 4.5: Sample positional encoding values for the sentence ”I bought an apple”

		”i” : 2	”bought” : 6	”an” : 3	”apple” : 0
i = 0	2i = 0	$\sin(0/10000^{0/4}) = 0$	0.8414	0.9093	0.1411
i = 0	2i+1 = 1	$\cos(0/10000^{0/4}) = 1$	0.5403	−0.4161	−0.9899
i = 1	2i = 2	$\sin(0/10000^{2/4}) = 0$	0.0099	0.0199	0.0299
i = 1	2i+1 = 3	$\cos(0/10000^{2/4}) = 1$	0.9999	0.9998	0.9995

We can observe how the values vary on different positions on different waves with wavelengths giving a distinct identity for each token at each position in below graphs fig4.9:



(a) 2i = 0

(b) 2i+1 = 1

(c) 2i = 2

(d) 2i+1 = 3

Figure 4.9: Visualizing sinusoidal positional encoding variations

Adding Word Embedding Matrix and Positional Embedding Matrix

This step consists of adding the values obtained after both word embedding and positional encoding, which caries information regarding tokens and their postions in the respective inputs they appeared in at each epoch. The process is described in Fig:4.10

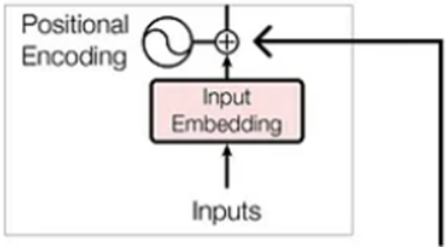


Figure 4.10: Summation Step

Table 4.6: Word Embedding Matrix

w1	w2	w3	w4
0.23	0.34	0.33	0.61
0.67	0.79	0.22	0.58
0.47	0.92	0.42	0.51
0.21	0.17	0.34	0.76

Table 4.7: Positional Embedding Matrix

p1	p2	p3	p4
0	0.8414	0.9093	0.1411
1	0.995	0.98	0.9553
0	0.01	0.0199	0.0299
1	1	0.9999	0.9999

$\Rightarrow$

Table 4.8: WP Matrix (Word + Positional Embedding)

wp1	wp2	wp3	wp4
0.23	1.1814	1.2393	0.7511
1.67	0.8895	1.2	1.5353
0.47	0.93	0.4399	0.5399
1.21	1.17	1.3399	1.7599

4.2.4 Attention Mechanism

Attention is a powerful mechanism in modern neural networks that enables models to dynamically focus on relevant parts of the input data when processing sequences. Attention

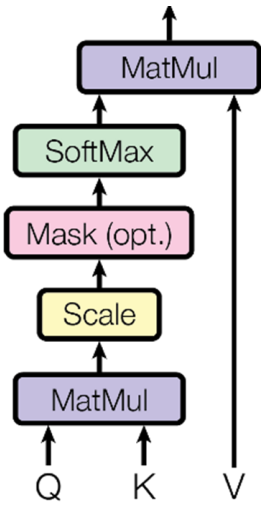


Figure 4.11: Self-attention

mainly comes in two forms: **self-attention**, where elements within the same sequence interact with each other, and **cross-attention**, where one sequence attends to another (commonly used in encoder-decoder models). The original Transformer model introduced in the paper *"Attention Is All You Need"* also proposed **multi-head attention** as **masked multi-head attention**, which runs multiple attention operations in parallel to capture different types of contextual information simultaneously.

Self-Attention

Self-attention, Fig:4.11, is a mechanism that captures dependencies and relationships within input sequences. It allows the model to identify and weigh the importance of different parts of the input sequence by attending to itself.

A transformer achieves this with the help of three special matrices:

- **(Q)**uery,
- **(K)**ey, and
- **(V)**alue.

The attention mechanism is computed as follows as shown in Fig:4.11:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

where $d_k = \frac{d_{\text{model}}}{h}$, with h being the number of attention heads, and d_{model} representing the model's hidden vector dimension.

Query and key matrices help us visualize the tokens in different embeddings. These matrices store information representing the similarity between tokens. The Q and K matrices are calculated as:

$$Q = (WP \text{ Matrix})^\top \times Q\text{-Weights Matrix}$$
$$(4 \times 4) \qquad (4 \times N)$$

$$K = (WP \text{ Matrix})^\top \times K\text{-Weights Matrix}$$
$$(4 \times 4) \qquad (4 \times N)$$

Here, the WP matrix is 4×4 dimensional based on the example sentence we used earlier, which had 4 tokens, and 4 is the d_{model}, the assumed embedding dimension. N can be any value.

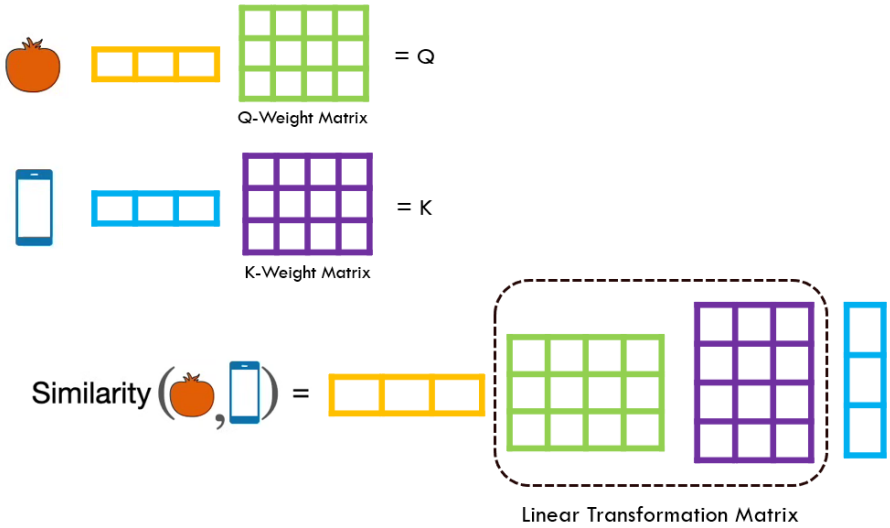


Figure 4.12: Linear Transformation matrix derived from Q and K

Q and K weight matrices act like transformation matrices that help us visualize the current coordinate system for tokens into another coordinate system that provides better separability among tokens, thereby helping the model to understand the input more effectively.As observed in Fig:4.12 we are modifying the similarity in one embedding by converting it to a different embedding. In this way Q and K matrices transform our embedding to a better suited embedding for this attention problem. Fig:4.13 Shows how similarity is measured in different transformations enhancing the dissimilarity between two tokens, which plays an important role in distinguishing between two tokens.

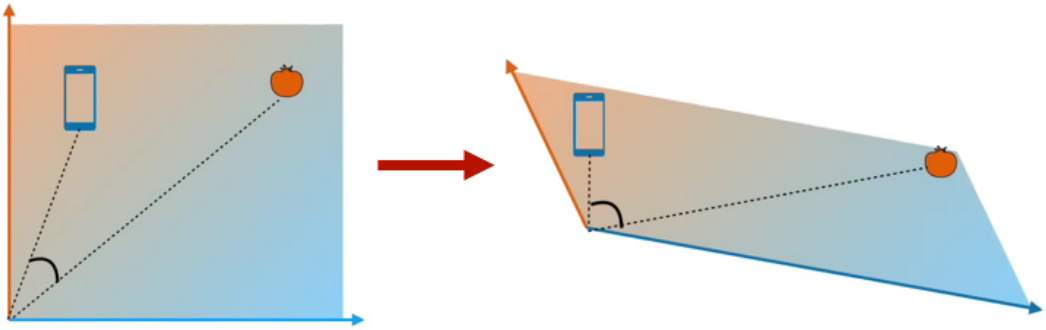


Figure 4.13: Visual representation of transformation

Value matrix (V) – While the Q and K matrices only find similarities between different features of the input, the V matrix helps in actually moving the word in question closer to those features. V is another linear transformation matrix, Fig:4.14 that maps words into an embedding that is best suited for next-word prediction.

$$V = (WP \text{ Matrix})^{\top} \times V\text{-Weights Matrix}$$
$$(4 \times 4) \qquad (4 \times N)$$

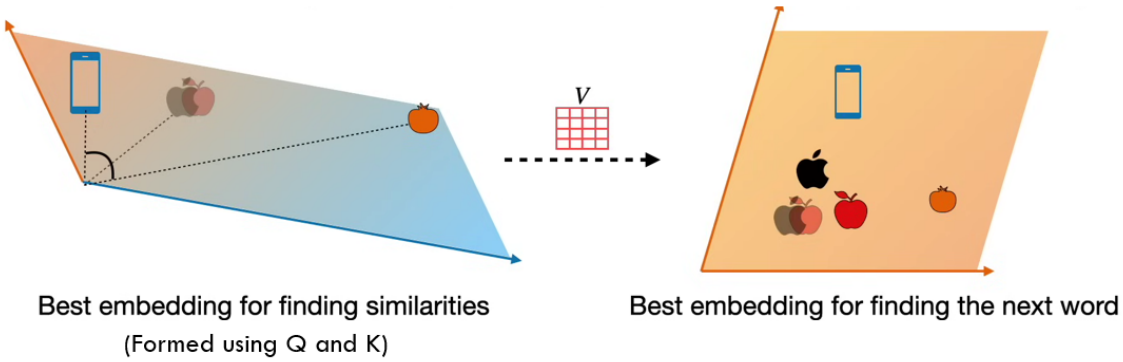


Figure 4.14: Transformation by V

Values in Q, K and V weight matrices are between -1 and 1 and are filled randomly in the beginning. They will later be updated as our transformer starts learning. Table:4.9

Cross-Attention

While self-attention focuses on capturing dependencies within the same sequence, **cross-attention** extends this concept by allowing the model to attend to a different input sequence. This is particularly useful in encoder-decoder architectures, such as in machine

Table 4.9: Comparison of Q, K, and V matrices

Q and K Matrices	**V Matrix**
Coordinates of words are not changed in the embedding	Coordinates for words are changed in the embedding
Does not modify original WP matrix	It modifies the original WP matrix
Used to find best similarity	Used to find best next word in the prediction

translation tasks, where the decoder uses cross-attention to focus on relevant parts of the encoder's output. Just like self-attention, cross-attention also relies on the Q, K, and V matrices, but with a key difference: the Q is derived from the decoder input, while K and V are computed from the encoder output. This enables the decoder to selectively extract information from the encoded source sequence to guide the generation of the output sequence.

$$
\begin{aligned}
Q &= (\text{Decoder Output})^T \times \text{Q-Weights Matrix} && (\text{Shape: } T_d \times d_k) \\
K &= (\text{Encoder Output})^T \times \text{K-Weights Matrix} && (\text{Shape: } T_e \times d_k) \\
V &= (\text{Encoder Output})^T \times \text{V-Weights Matrix} && (\text{Shape: } T_e \times d_v)
\end{aligned}
$$

Here, T_d is the number of tokens in the decoder input, T_e is the number of tokens in the encoder output, and d_k, d_v are the dimensions of the key and value vectors respectively. This formulation allows the decoder to attend over encoder representations and generate contextually rich outputs.

Multi-head Attention

The mechanism that enables the Transformer to achieve parallelism and capture diverse aspects of the input representation is known as **Multi-Head Attention**, refer Fig:4.15. Instead of applying a single self-attention function, the Transformer uses multiple attention heads—each with its own set of learnable weight matrices W_Q^i, W_K^i, and W_V^i, as seen in Fig:4.16. These heads act like independent *workers*, each focusing on different subspaces or relationships within the sequence.

Here multiple single head blocks work in parallel

Mathematically, the output of multi-head attention is given by:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where each head is defined as:

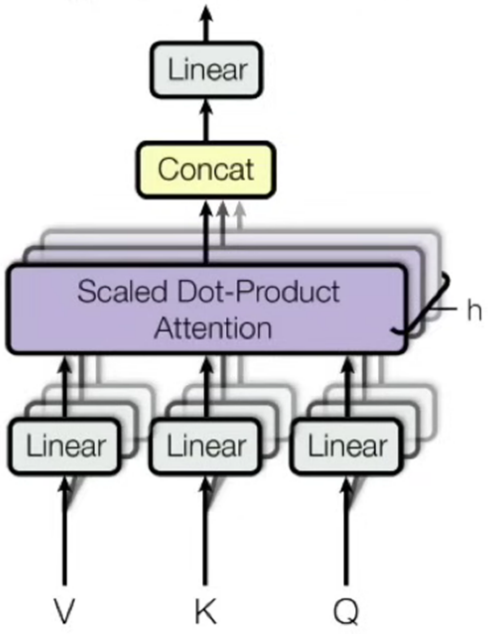


Figure 4.15: Multi-Head Attention

$$\text{head}_i = \text{Attention}(QW_Q^i, KW_K^i, VW_V^i)$$

Here:

- Q, K, V are the input query, key, and value matrices.
- W_Q^i, W_K^i, and W_V^i are the learnable projection matrices for the i-th head.
- W^O is the output projection matrix applied after concatenation.
- h is the number of attention heads.

The concatenated output has a dimensionality of $d_k \times (N \cdot h)$, where N is the length of the input sequence and d_k is the dimensionality of each head. Since the original input to the multi-head attention layer has dimensionality $d_k \times$ (number of tokens), an additional linear transformation (via W^O) is applied to map the concatenated output back to the expected shape.

This is accomplished by applying a linear transformation using the matrix W^O. The values in this matrix are initialized randomly, typically with values between 0 and 1, and are updated during the training process as the Transformer learns.

Concatenation happens along the x-axis like Figure:4.17:

This design allows the model to jointly attend to information from different representation subspaces at different positions. Each attention head computes attention in parallel,

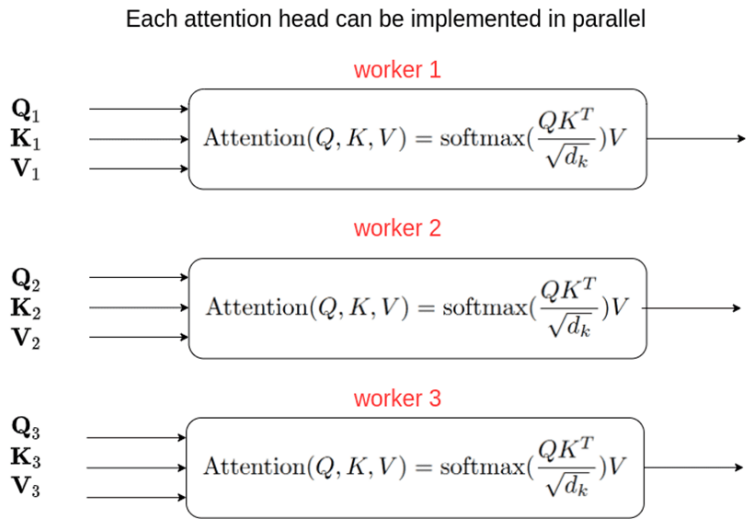


Figure 4.16: Multiple single head attentions

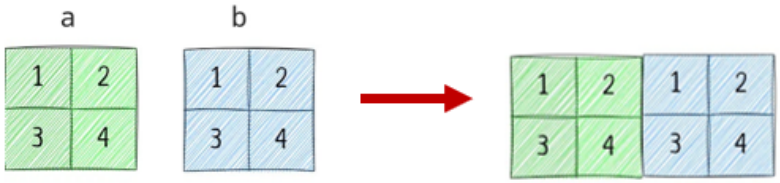


Figure 4.17: Concatenation

and their outputs are then concatenated and projected using a final linear layer to obtain the final representation. Refer: 4.19

Output of Multi-Head Attention with One Head

Multi-head attention matrix is calculated by applying the attention formula over multiple workers (or heads), Fig:4.18 and their outputs are concatenated. However, since we are considering only one worker (head), the output dimension remains the same — hence, no concatenation is needed.

Let h = number of attention heads = 1

Matrix Multiplication: Attention Output × Linear Weights

$$\begin{bmatrix} 2.073 & 1.937 & 3.082 \\ 2.096 & 1.951 & 3.126 \\ 2.089 & 1.947 & 3.113 \\ 2.092 & 1.949 & 3.119 \end{bmatrix} \times \begin{bmatrix} 0.34 & 0.12 & 0.78 & 0.45 \\ 0.28 & 0.71 & 0.66 & 0.69 \\ 0.32 & 0.41 & 0.91 & 0.53 \end{bmatrix} = \begin{bmatrix} 2.234 & 2.888 & 5.700 & 3.903 \\ 2.259 & 2.918 & 4.767 & 3.946 \\ 2.252 & 2.909 & 5.748 & 3.934 \\ 2.255 & 2.913 & 5.756 & 2.939 \end{bmatrix}$$

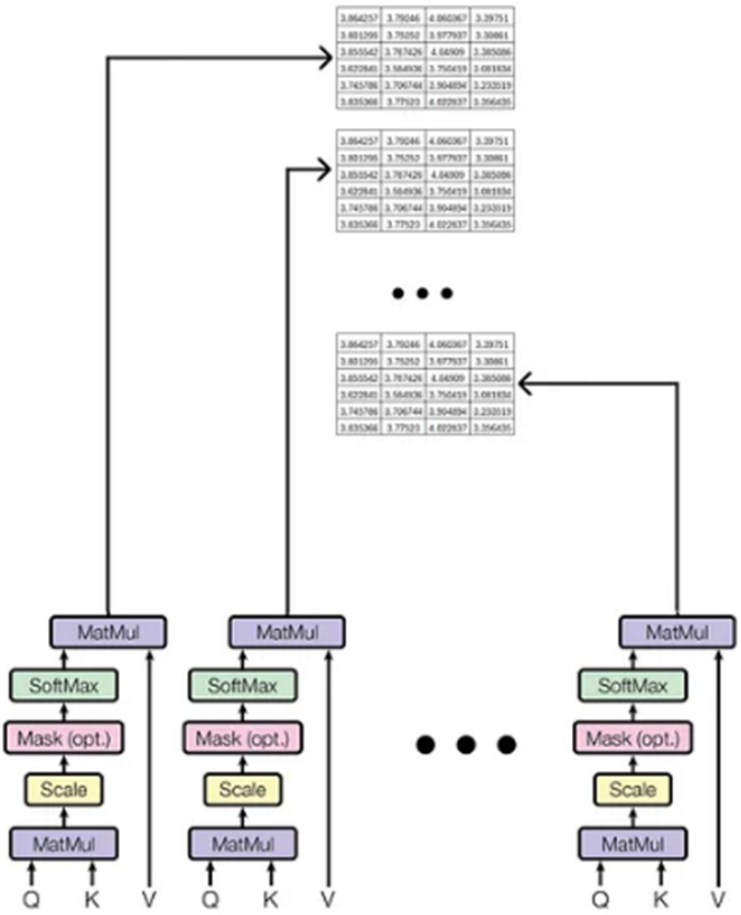


Figure 4.18: Multi-head concatenation

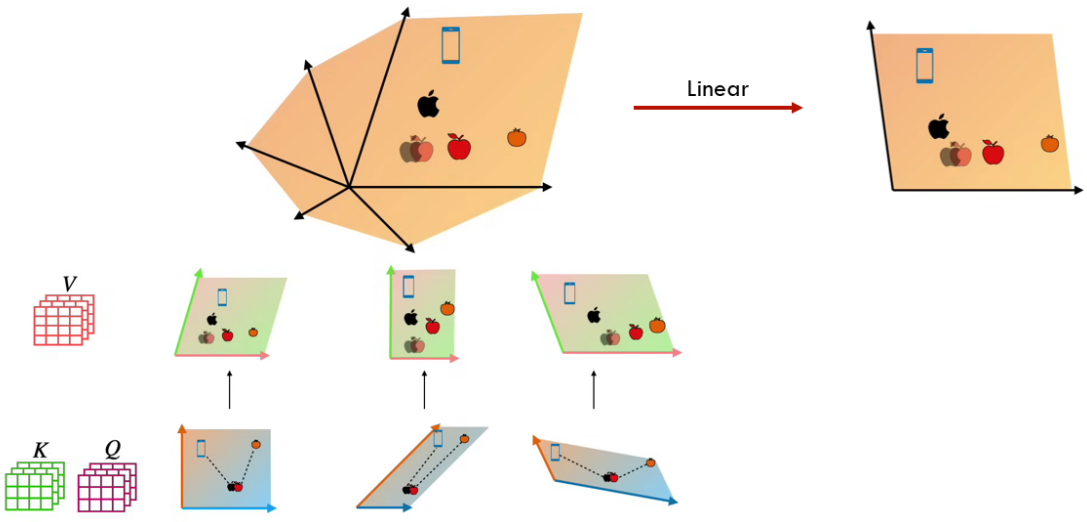


Figure 4.19: Multi-head Attention output

Note

Values in the Linear matrix are between -1 and 1 and are filled randomly in the beginning. They will later be updated as the transformer trains.

Masked Multi-head Attention

This is the first attention layer in decoder layer. This is nothing but a multi-head attention but with masking of some values. This attention is a self-attention where attention score between decoder tokens are calculated and establish relationship between them. While predicting one token is fed to decoder at a time and hence to mask the future possible values being predicted before only the tokens next to present token are masked. This way model doesn't predicts next tokens before only and learn token by token of the target sentence.

Summary on various layers of attentions

Refer to 4.10 that is a summarizing table about the attention layer used in encoder and decoder:

Table 4.10: Comparison of Multi-head Attention Types in Transformer Architecture

Parameters	**Multi-head Attention in Encoder**	**Masked Multi-head Attention in Decoder**	**Multi-head Attention in Decoder (Cross Attention)**
Location	Encoder block	Decoder block	Decoder block
Attention Type	Self attention	Self attention	Cross attention
Inputs	Source tokens	Target tokens	Source–target token pairs
Outputs	Attention between source tokens	Attention between target tokens	Attention between source and target tokens
Masking	Absent	Present	Absent
Q, K, V	Q, K, V come from the output of previous encoder block	Q, K, V come from the output of previous decoder block	Q comes from previous decoder layer, K and V come from the encoder output
Formula	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T+M}{\sqrt{d_k}}\right)V$	$\text{Attn}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

4.2.5 Adding and Normalization

The output of multi-head attention is added with the transpose of the initial weight matrix obtained after summation of encoding and embedding matrix (denoted as WP^T). Then

the result is normalized row-wise using standardization to ensure that weights are scaled to avoid gradient diminishing or exploding issues during training.

Matrix Addition

$$WP^T = \begin{bmatrix} 0.23 & 1.67 & 0.47 & 1.21 \\ 1.1814 & 0.8895 & 0.93 & 1.17 \\ 1.2393 & 1.2 & 0.4399 & 1.3399 \\ 0.7511 & 1.5353 & 0.5399 & 1.7599 \end{bmatrix} + \begin{bmatrix} 2.234 & 2.888 & 5.700 & 3.903 \\ 2.259 & 2.918 & 4.767 & 3.946 \\ 2.252 & 2.909 & 5.748 & 3.934 \\ 2.255 & 2.913 & 5.756 & 2.939 \end{bmatrix}$$

$$= \begin{bmatrix} 2.464 & 4.558 & 6.170 & 5.113 \\ 3.440 & 3.807 & 5.697 & 5.116 \\ 3.491 & 4.109 & 6.188 & 5.274 \\ 3.006 & 4.448 & 6.296 & 4.699 \end{bmatrix}$$

Row-wise Standardization

$$\text{StandardScaler}() \Rightarrow \begin{bmatrix} -1.564 & -0.135 & 1.180 & 0.3975 \\ -1.163 & -0.766 & 1.279 & 0.650 \\ -1.224 & -0.630 & 1.366 & 0.488 \\ -1.376 & -0.140 & 1.442 & 0.074 \end{bmatrix}$$

4.2.6 Feed Forward Neural Network

After normalizing the matrix, it will be processed through a feed-forward network Fig:4.21. For this example, we will be using a very basic network that contains only one linear layer and one ReLU activation function layer as Fig:4.20.

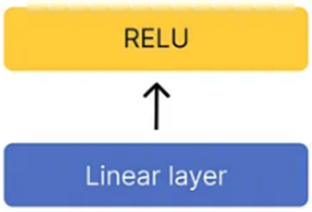


Figure 4.20: FFN Layers

The feed-forward network performs two main tasks:

1. **Non-linear Transformation:** Self-attention provides context-aware representations of the input. The feed-forward network adds another layer of complexity by applying non-linear functions to these representations. This allows the model to learn more intricate patterns in the data.

2. **Feature Enhancement:** The feed-forward network can also be seen as a way to refine the features extracted by the self-attention layer. It can emphasize important features and suppress less relevant ones, making the final output more informative.

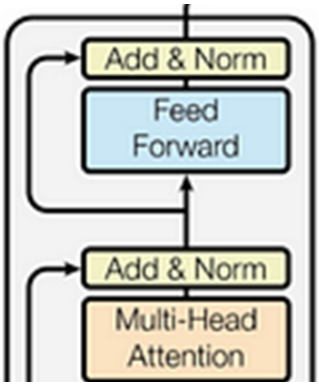


Figure 4.21: Feed Forward Neural Network

4.2.7 Decoder

As seen in the Transformer architecture, Fig:4.22 the decoder follows a similar flow as the encoder, but it involves the calculation of two attention matrices:

- **Masked Multi-head Attention:** This is self-attention applied on the target tokens.
- **Multi-head Attention:** This is cross-attention between the input (source) and target tokens.

When training, there are two inputs to the decoder:

1. The first input comes from the encoder. The output matrix of the last `Add & Norm` layer in the encoder serves as the **Key** and **Value** for the second multi-head attention layer in the decoder. The **Query** matrix comes from the decoder after the first `Add & Norm` step, which follows the first (masked) multi-head attention layer.
2. The second input is the embedding of the predicted text, which represents the target tokens.

Here, the **Value** matrix contains embeddings corresponding to the target tokens. When we perform attention using the **Query** and **Key** matrices from the encoder, the decoder attempts to map the output tokens according to the input tokens. This mechanism enables the model to establish relationships between the source and target tokens.

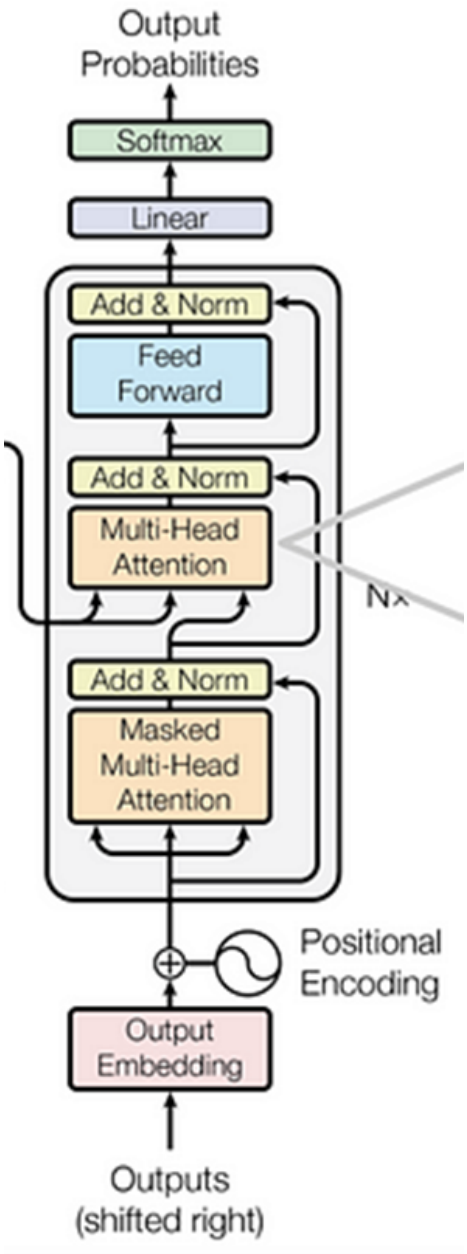


Figure 4.22: Decoder

This is how **cross-attention** works: the output tokens are projected using a linear transformation derived from the input tokens. These target tokens thereby learn to relate not only among themselves but also with the source sequence.

The final output of the decoder, after passing through the last `Add & Norm` block, is then flattened to match the input requirements of a linear layer. This linear layer is responsible for predicting the next word by computing logits:

$$\text{Flattened Layer} \times \text{Linear Weights} = \text{Logits}$$

Let this matrix be the matrix obtained after the last **Add & Norm** step:

$$\begin{bmatrix} 1.20 & 1.33 & 2.11 & 2.63 \\ 0.56 & 1.45 & 2.24 & 2.78 \\ 3.56 & 3.32 & 3.64 & 4.55 \\ 1.04 & 0.91 & 1.92 & 1.70 \end{bmatrix}$$

Let this matrix be the matrix obtained after the last **Add & Norm** step:

$$\begin{bmatrix} 1.20 & 1.33 & 2.11 & 2.63 \\ 0.56 & 1.45 & 2.24 & 2.78 \\ 3.56 & 3.32 & 3.64 & 4.55 \\ 1.04 & 0.91 & 1.92 & 1.70 \end{bmatrix}$$

This matrix is then **flattened** in row-major order into a 1×16 vector:

$$X = \begin{bmatrix} 1.20 & 1.33 & 2.11 & 2.63 & 0.56 & 1.45 & 2.24 & 2.78 & 3.56 & 3.32 \\ 3.64 & 4.55 & 1.04 & 0.91 & 1.92 & 1.70 \end{bmatrix}$$

Shape: 1×16

$$\textbf{Linear Layer} = X \cdot W \quad \text{(No Bias)}$$

Flattened layer:

$$X = \begin{bmatrix} 1.2 & 1.33 & \cdots & 1.92 & 1.7 \end{bmatrix} \quad \text{(Shape: } 1 \times 16\text{)}$$

Linear Weights:

$$W = \begin{bmatrix} 0.72 & 0.17 & 0.62 & \cdots & 0.54 \\ 0.14 & 0.23 & 0.58 & \cdots & 0.11 \\ 0.50 & 0.93 & 0.47 & \cdots & 0.05 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.16 & 0.02 & 0.67 & \cdots & 0.87 \end{bmatrix} \quad \text{(Shape: } 16 \times 10\text{, where 10 is the vocab size)}$$

Logits (output of the linear layer):

$$\text{logits} = \begin{bmatrix} 1.14 & 5.4 & 1.15 & \cdots & 3.3 \end{bmatrix} \quad \text{(Shape: } 1 \times 10\text{)}$$

Token indices:

apple:0	phone:1	i:2	···	orange:9
1.14	5.4	1.15	···	3.3

Here, the **logits** represent a row vector containing unnormalized scores for each token in the dataset (corpus).

Finally, a **softmax** function is applied to convert the logits into probabilities. The token with the highest probability is selected as the predicted output of the model.

$$\textbf{logits} = \begin{bmatrix} \text{apple:0} & \text{phone:1} & \text{i:2} & \cdots & \text{orange:9} \\ 1.14 & 5.4 & 1.15 & \cdots & 3.3 \end{bmatrix}$$

$$\xrightarrow{\text{Softmax}}$$

$$\textbf{probabilities} = \begin{bmatrix} \text{apple:0} & \text{phone:1} & \text{i:2} & \cdots & \text{orange:9} \\ 0.21 & \mathbf{0.57} & 0.09 & \cdots & 0.33 \end{bmatrix}$$

Highest probability $\rightarrow$ **phone:1**

Based on our calculations, the predicted word is ``phone''. This will now act as an input to the decoder in the next iteration. This process will continue until the <end> token is predicted.

4.3 Transformer Dimension Flow Summary

Variables

- t: context window = 4
- d_{model}: dimensionality = 4
- $d_k = d_v$: weight matrix columns = 3
- h: number of attention heads = 1

Steps

1. Word embedding matrix W, positional encoding matrix P, combined matrix $WP = W + P$:

$$WP \in \mathbb{R}^{d_{\text{model}} \times t} = \mathbb{R}^{4 \times 4}$$

2. Transpose of combined matrix:

$$WP^T \in \mathbb{R}^{t \times d_{\text{model}}} = \mathbb{R}^{4 \times 4}$$

3. Query, Key, and Value weight matrices:

$$W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d_k} = \mathbb{R}^{4 \times 3}$$

4. Computing Q, K, V:

$$Q = WP^T \cdot W_Q, \quad K = WP^T \cdot W_K, \quad V = WP^T \cdot W_V$$

$$Q, K, V \in \mathbb{R}^{t \times d_k} = \mathbb{R}^{4 \times 3}$$

5. Scaled Dot-Product Attention:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V$$

$$\text{Output} \in \mathbb{R}^{t \times d_v} = \mathbb{R}^{4 \times 3}$$

6. Concatenated multi-head attention output:

$$\text{Concat}(head_1, \ldots, head_h) \in \mathbb{R}^{t \times (d_v \cdot h)} = \mathbb{R}^{4 \times 3}$$

7. Linear transformation weight matrix:

$$W_O \in \mathbb{R}^{(d_v \cdot h) \times d_{\text{model}}} = \mathbb{R}^{3 \times 4}$$

8. Output of multi-head attention layer:

$$\text{MultiHeadOutput} = \text{Concat}(...) \cdot W_O \in \mathbb{R}^{t \times d_{\text{model}}} = \mathbb{R}^{4 \times 4}$$

9. Adding and normalization:

$$\text{Norm}(WP^T + \text{MultiHeadOutput}) \in \mathbb{R}^{t \times d_{\text{model}}} = \mathbb{R}^{4 \times 4}$$

10. Feed-forward output:

$$\text{FFN Output} \in \mathbb{R}^{t \times d_{\text{model}}} = \mathbb{R}^{4 \times 4}$$

Chapter 5

ANALYSIS AND VISUALIZATION OF ATTENTION

We would be discussing through various data visualization techniques like: Heat maps, Graphs, Clusters and line graphs to study the effect of attention through different layers and heads. This visualization would enable us to prove that how attention is working in developing relationships between tokens, while keeping track of all the information derived from sentences which they have been trained on. For examples we are considering the sentences trained on the base model along with their saved attention scores among each token.

5.1 Heat Maps

Aim: Using heatmap to observe how attention weights changes between tokens at various layer, signifying tokens learning at various layers and heads.
Concept: Here we have constructed heat map with the help of attention weights and tokens saved after training the model for 30 epochs. The heat maps are formed by plotting attention weights between any tokens given as input to encoder, decoder and encoder-decoder layer of the transformer. In above images we can see heat maps for encoder and decoder for layer 5 and head 0. The darker the color indicates higher the attention weight between those 2 tokens. Refer to Algo:1 for algortihm.

Example: Refer to Fig:5.1

Source: This was the question.
Target: यह प्रश्न था।

Figure 5.1: Heat Map example

Observation derived: Referring to Fig. 5.2, the heatmaps of the initial layers typically show darker boxes along the diagonal, indicating that a token is most similar to itself in both the encoder and decoder layers.

Algorithm 1 Attention Map Visualization using Heat Map

Require: attn_type, layers, heads, row_tokens, col_tokens, max_sentence_len

charts ← empty list
for all layer ∈ layers **do**
    rowCharts ← empty list
    for all head ∈ heads **do**
        attn ← GET_ATTENTION(attn_type, layer, head)
        data_list ← empty list
        for $r \leftarrow 0$ to max_sentence_len-1 **do**
            for $c \leftarrow 0$ to max_sentence_len-1 **do**
                value ← attn[r][c]
                row_token ← (if $r <$ |row_tokens| then row_tokens[r] else "blank ")
                col_token ← (if $c <$ |col_tokens| then col_tokens[c] else "blank ")
                append $(r, c, value,$ row_token, col_token$)$ to data_list
        df ← DataFrame(data_list)
        chart ← DRAW_HEATMAP(df)
return

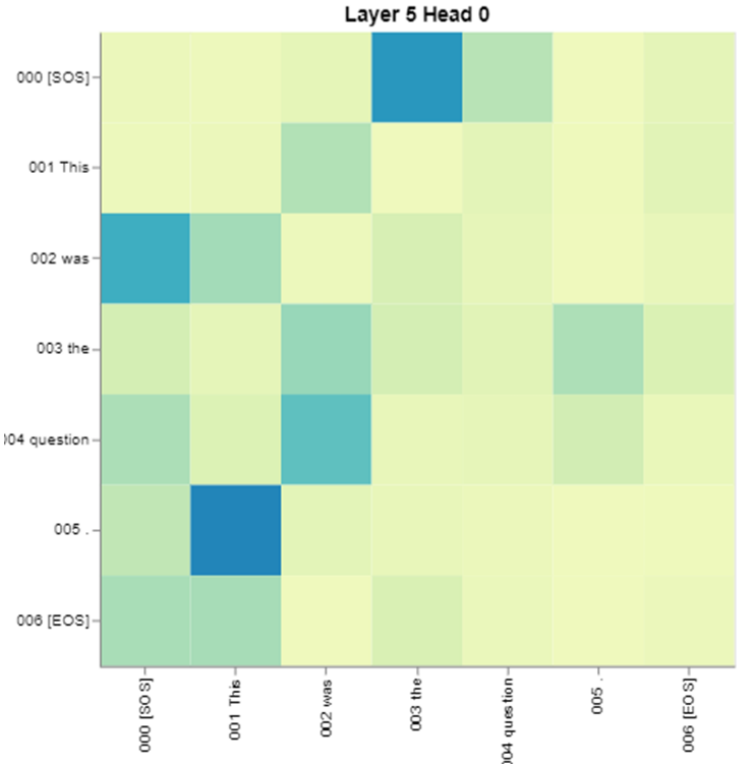
- Heatmaps of initial layers usually have darker boxes on the diagonal, highlighting that a token is most similar to itself, for both the encoder and decoder layers.
- The case differs for the encoder-decoder attention layer, as the encoder tokens are in English and the decoder tokens are in Hindi. Hence, the attention is computed based on the learned weight matrices between the two languages.
- In later layers, we can observe that certain words begin to establish stronger relationships with each other, based on contextual dependencies captured by the model.

5.2 Graphs

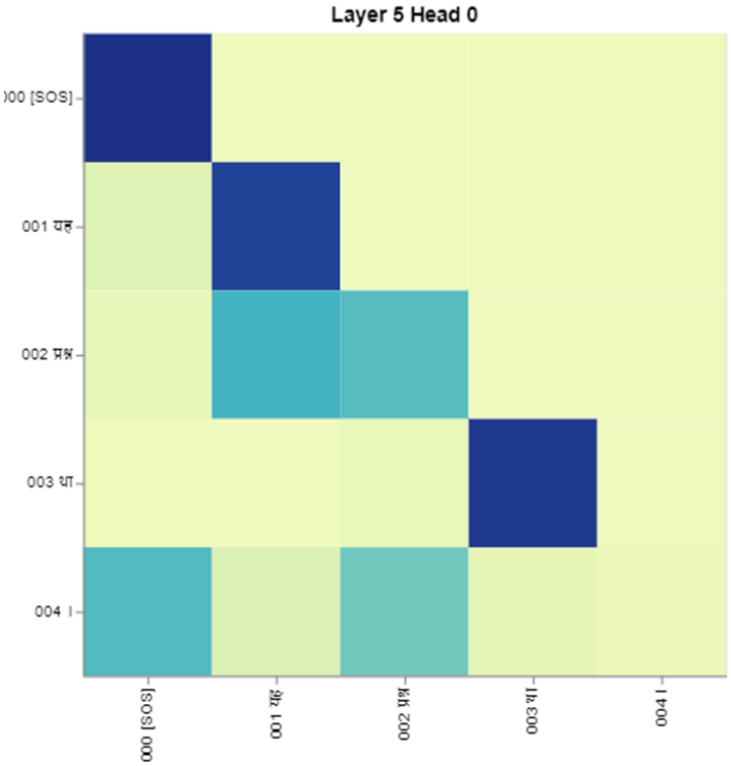
Aim: To detect and observe how tokens attract other tokens across epochs and using these graphs ahead construct clusters of similar tokens.
Concept: Graph analysis is used here to visualize token similarity and relational paths learned by the model. These graphs help understand the internal reasoning structure across layers, fulfilling the interpretability goal of the study. Tokens of dataset are considered as nodes whereas the attention weights are considered as edges between any two nodes. Using the same we get undirected graphs. Refer to Algo:2 for algortihm.

Continuing the example in Heat Maps visualisation, the graphs are constructed as Fig:5.3:

Observation derived: Referring to Fig. 5.3



(a) Encoder(Layer 5 Head 0)



(b) Decoder(Layer 5 Head 0)

Figure 5.2: Heat Map Visualisation

Algorithm 2 Visualisation using graph

Require: attn_type, layer, head, threshold

attn_matrix ← GET_ATTENTION(attn_type, layer, head)
tokens ← CHOOSE_TOKENS(attn_type)
G ← INIT_GRAPH()
for $i = 0$ to rows(attn_matrix)-1 **do**
    for $j = 0$ to cols(attn_matrix)-1 **do**
        if attn_matrix[i][j] $>$ threshold **then**
            ADD_EDGE(G, tokens[i], tokens[j], attn_matrix[i][j])
PLOT_GRAPH(G)

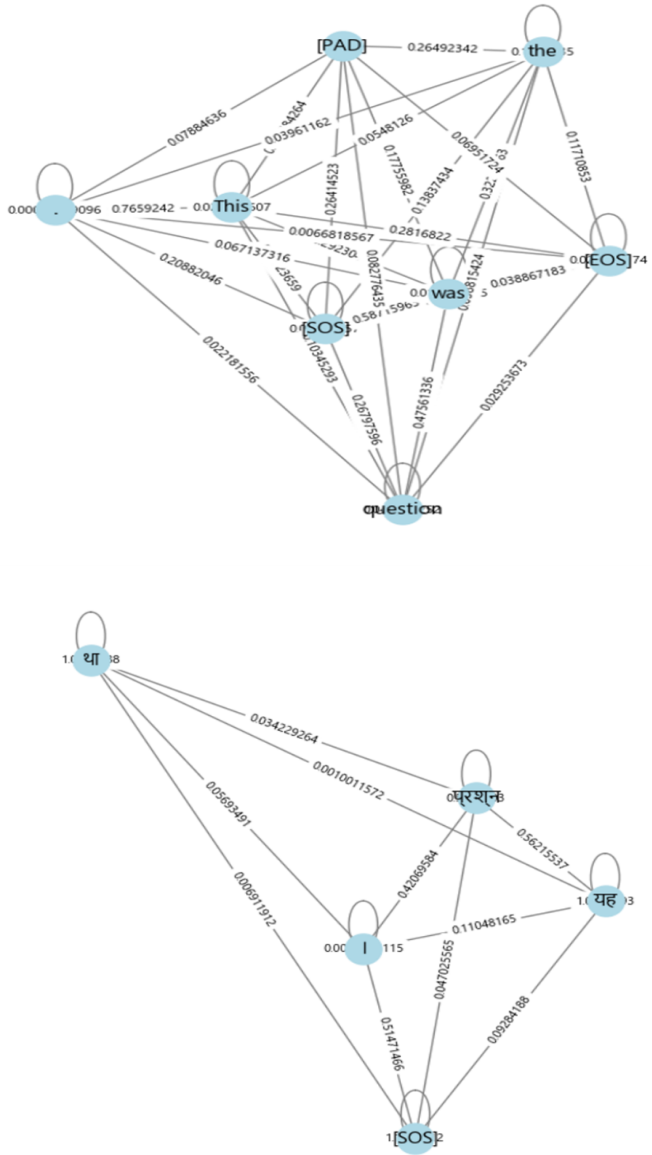


Figure 5.3: Graphical Visualisation at Layer 5 Head 0

- Tokens try to map similar meaning while decoding after learning from encoder step from input tokens.
- Usually words like nouns, verbs and adjectives are the meaning driven words around which the relationship is woven.

5.3 Cluster

Aim: We will be covering here three types of analysis:

1. **Detecting Communities (Clusters):** To represent nodes in clusters showing association of tokens closest to each other. This helps us detect how the model is grouping tokens at each step.
2. **Betweenness Centrality:** To detect the leader of those clusters, or the tokens which are drivers of a particular sentence. This also validates whether the clusters are rightly formed or not.
3. **Top-*k* Important Nodes:** To find and represent key tokens that the model relies on to propagate information across the sequence. Highlighting these tokens can help explain why the model attends to certain tokens more than others.

5.3.1 Detecting communities (clusters):

Concept: Building communities

- We are achieving this by creating graph clusters using the `perform_clustering()` function from the `network` package in Python.
- The `perform_clustering` function uses the Louvain method (`greedy_modularity_communities`) to detect communities (clusters) in the graph. Each community represents a group of nodes that are more densely connected to each other than to the rest of the graph.
- The Louvain method is a popular algorithm for detecting communities (clusters) in networks by optimizing *modularity*, a metric that quantifies the quality of a partition of a network into communities.

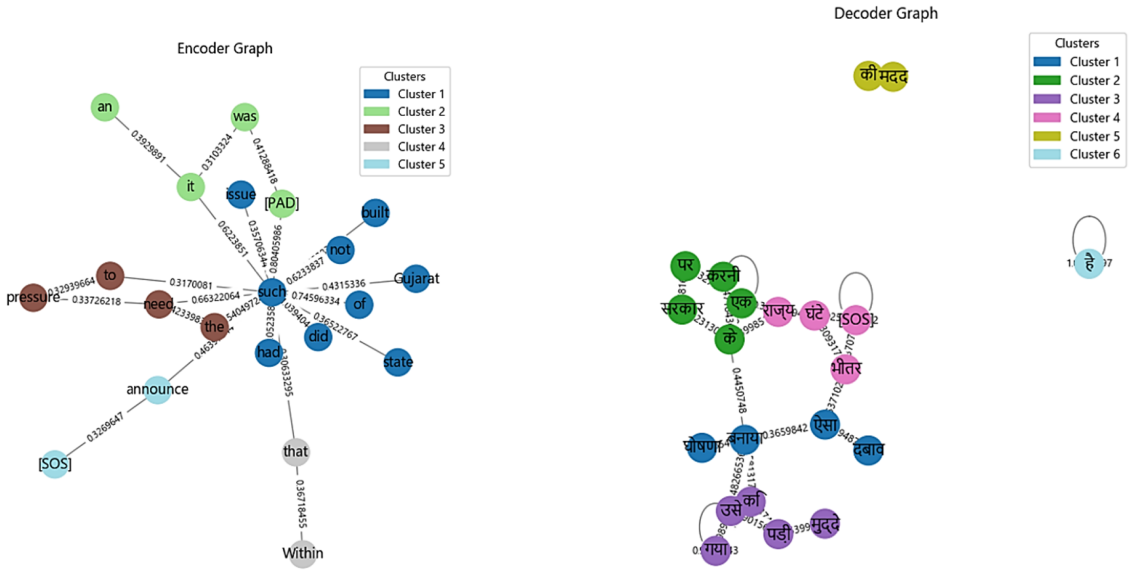
Example: Refer to Fig:5.4

Observations Derived

Source: Within an hour such a pressure was built on the state government that it had to announce that it did not need help of Gujarat to deal with the issue.

Target: एक घंटे के भीतर राज्य सरकार पर ऐसा दबाव बनाया गया कि उसे यह घोषणा करनी पड़ी कि उसे इस मुद्दे से निपटने के लिए गुजरात की मदद की जरूरत नहीं है।

Figure 5.4: Cluster Analysis Example



(a) Clusters in encoder(Layer 5 Head 0) **(b) Clusters in decoder(Layer 5 Head 0)**

Figure 5.5: Clusters

- Clusters are representing semantic groups (e.g., nouns, verbs, or phrases that belong to the same context) but also include a few prepositions alongside.
- These clusters become more segregated according to semantics in the later layers.
- Clusters help identify attention patterns (e.g., which tokens the model focuses on together).
- Input and target tokens may not have the same number of clusters.
- Also, the clusters are not synonymous with each other, indicating that the model is not memorizing (byhearting) but rather learning.

5.3.2 Betweenness Centrality:

Concept: Building communities Betweenness centrality is a measure of a node's importance in a graph based on how often it lies on the shortest path between other nodes.

- Clusters are sorted by an importance score calculated by summing the betweenness centrality scores of all nodes present in each cluster.
- These clusters may change from layer to layer as attention patterns evolve.

The formula for Betweenness Centrality is given as:

$$\text{Betweenness Centrality}(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5.1}$$

Where:

- $\sigma_{st}(v)$: Number of shortest paths from node s to node t that pass through node v
- σ_{st}: Total number of shortest paths from node s to node t

Continuing with the previous example in detecting communities, we get output as shown in Fig:5.6:

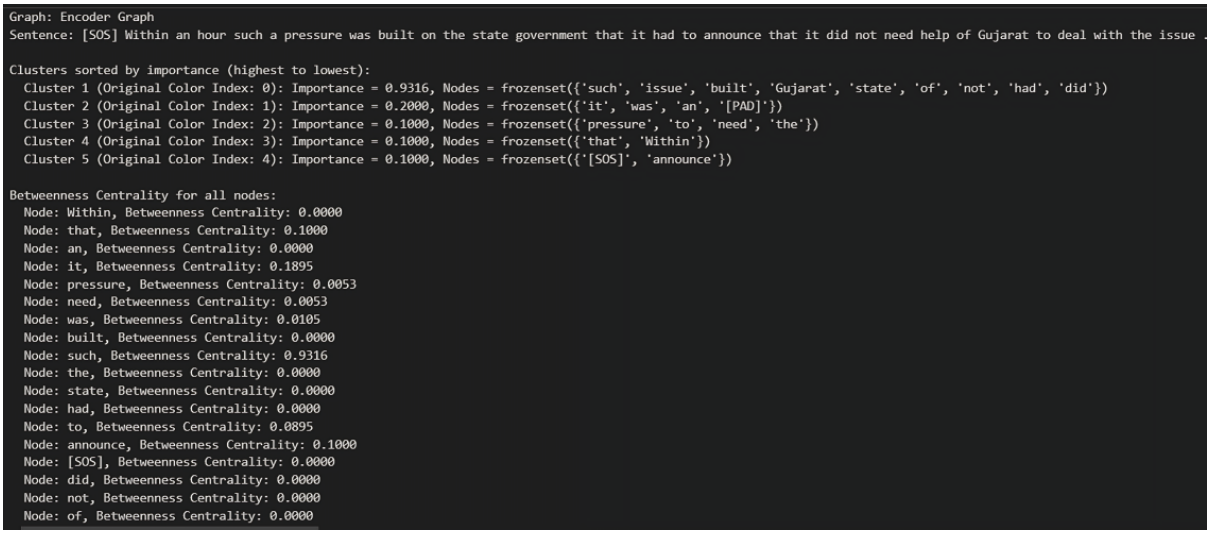

Graph: Encoder Graph
Sentence: [SOS] Within an hour such a pressure was built on the state government that it had to announce that it did not need help of Gujarat to deal with the issue
Clusters sorted by importance (highest to lowest):
Cluster 1 (Original Color Index: 0): Importance = 0.9316, Nodes = frozenset({'such', 'issue', 'built', 'Gujarat', 'state', 'of', 'not', 'had', 'did'})
Cluster 2 (Original Color Index: 1): Importance = 0.2000, Nodes = frozenset({'it', 'was', 'an', '[PAD]'})
Cluster 3 (Original Color Index: 2): Importance = 0.1000, Nodes = frozenset({'pressure', 'to', 'need', 'the'})
Cluster 4 (Original Color Index: 3): Importance = 0.1000, Nodes = frozenset({'that', 'Within'})
Cluster 5 (Original Color Index: 4): Importance = 0.1000, Nodes = frozenset({'[SOS]', 'announce'})
Betweenness Centrality for all nodes:
Node: Within, Betweenness Centrality: 0.0000
Node: that, Betweenness Centrality: 0.1000
Node: an, Betweenness Centrality: 0.0000
Node: it, Betweenness Centrality: 0.1895
Node: pressure, Betweenness Centrality: 0.0053
Node: need, Betweenness Centrality: 0.0053
Node: was, Betweenness Centrality: 0.0105
Node: built, Betweenness Centrality: 0.0000
Node: such, Betweenness Centrality: 0.9316
Node: the, Betweenness Centrality: 0.0000
Node: state, Betweenness Centrality: 0.0000
Node: had, Betweenness Centrality: 0.0000
Node: to, Betweenness Centrality: 0.0895
Node: announce, Betweenness Centrality: 0.1000
Node: [SOS], Betweenness Centrality: 0.0000
Node: did, Betweenness Centrality: 0.0000
Node: not, Betweenness Centrality: 0.0000
Node: of, Betweenness Centrality: 0.0000

Figure 5.6: Betweeness Centrality

Observations Derived

- A node with high betweenness centrality acts as a bridge or connector between different parts of the graph.
- It indicates how much influence a node has over the flow of information or relationships in the network.

5.3.3 Top-k Important Nodes:

Concept: The top-k nodes with the highest betweenness centrality are printed.
For example: sentence: Input Sentence: "Within an hour such a pressure was built on the state government that it had to announce that it did not need help of Gujarat to deal with the issue."

Clusters Sorted by Importance (Highest to Lowest):

1. **Cluster 1**: Importance = 0.9316, Nodes = {`such, issue, built, Gujarat, state, of, not, had, did'`}
2. **Cluster 2**: Importance = 0.2000, Nodes = {`it, was, an, [PAD]`}
3. **Cluster 3**: Importance = 0.1000, Nodes = {`pressure, to, need, the`}
4. **Cluster 4**: Importance = 0.1000, Nodes = {`that, Within`}
5. **Cluster 5**: Importance = 0.1000, Nodes = {`[SOS], announce'`}

Observations Derived: Top-k Nodes: {"such", "it", "that", "announce", "to"}

- **Interpretation:** Top nodes are the cluster heads that are pulling words and making clusters around them. This signifies that top heads are driving the flow of information among the edges and nodes.

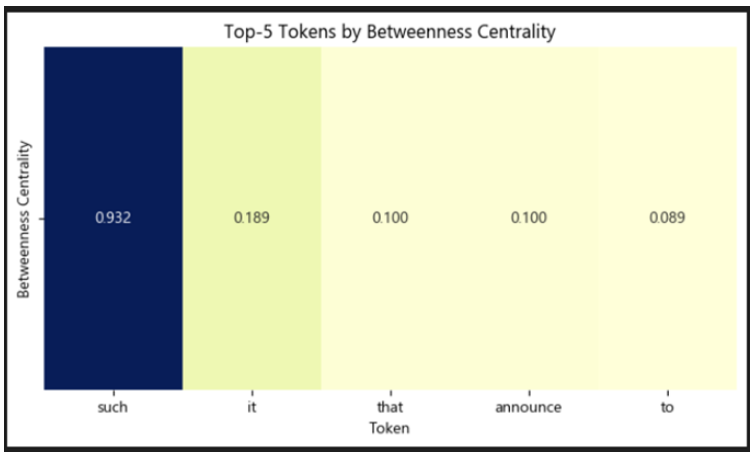
Combined algorithm for all 3 above methods : Algo:3

5.4 BLEU Score Analysis

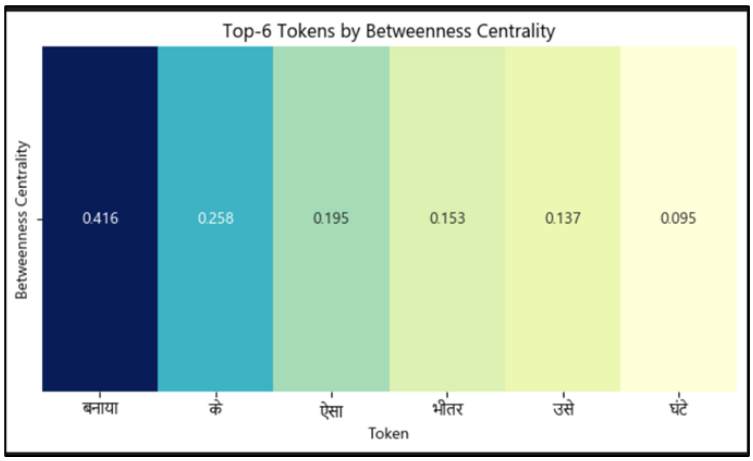
Aim: To calculate BLEU score for a given prediction at every epoch which will prove that model is learning, and will conclude how many epochs does model need generally. When does it start learning i.e. at which epoch the model starts building meaningful relationships.
Concept: The acronym **BLEU** stands for "Bilingual Evaluation Understudy," and it is a metric used to measure the accuracy of machine translations by comparing the machine-generated translations with human translations. Refer to Algo:4
The **formula** for BLEU score is given as:

$$\text{BLEU} = BP \times \exp\left(\sum (w_n \times \log p_n)\right)$$

where:



(a) Encoder



(b) Decoder

Figure 5.7: Important Nodes in Encoder and Decoder

- w_n are the weights assigned to each n-gram precision,
- p_n is the precision of n-grams (clipped count of matching n-grams) divided by the total number of n-grams in the candidate translation.

Example: Refer to Fig:5.1

Observations Derived

1. **Refering to figure: 5.8a**
    - The BLEU score keeps fluctuating between epochs, but as epochs increase, the BLEU score gradually settles towards a value of 1, confirming that the model is learning.
    - The fluctuation indicates that the model is trying different possible ways to generate the next word, based on the previous word.

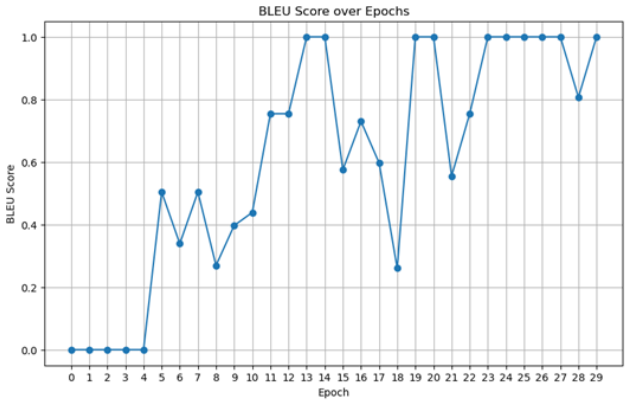
Algorithm 3 Analyze Attention Graphs With Clustering

Require: configurations $\subseteq \{(\text{attn\_type}, \text{layer}, \text{head})\}$, threshold, k
Ensure: Top-k nodes and colored cluster plots for each configuration

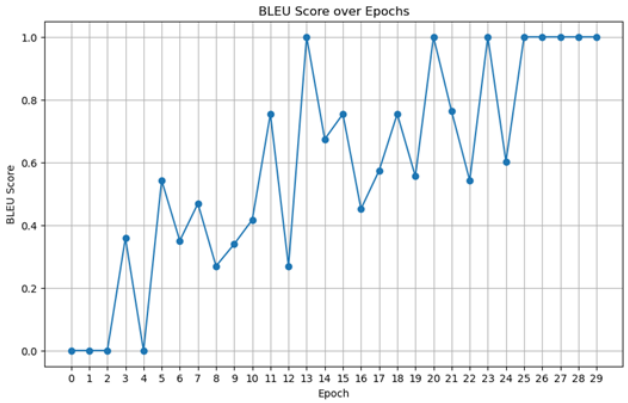
for all (t, ℓ, h) in configurations **do**
  ▷ 1. Build attention graph
  $G \leftarrow \text{matrix\_to\_graph}(t, \ell, h, \text{threshold})$
  ▷ 2. Detect communities (clusters)
  $\mathcal{C} \leftarrow \text{perform\_clustering}(G)$
  ▷ 3. Highlight clusters
  $\text{colors} \leftarrow \text{highlight\_clusters}(G, \mathcal{C})$
  ▷ 4. Compute betweenness centrality
  $B \leftarrow \text{calculate\_betweenness\_centrality}(G)$
  ▷ 5. Extract top-k nodes
  $\text{top\_nodes} \leftarrow \text{extract\_top\_k\_nodes}(B, k)$
  ▷ 6. Plot and print results
  plot_graph(G, colors)
  print "Config:", t, "Layer:", ℓ, "Head:", h
  print "Top-", k, " nodes:", top_nodes

Algorithm 4 Evaluate Model Across Epochs using BLEU Score

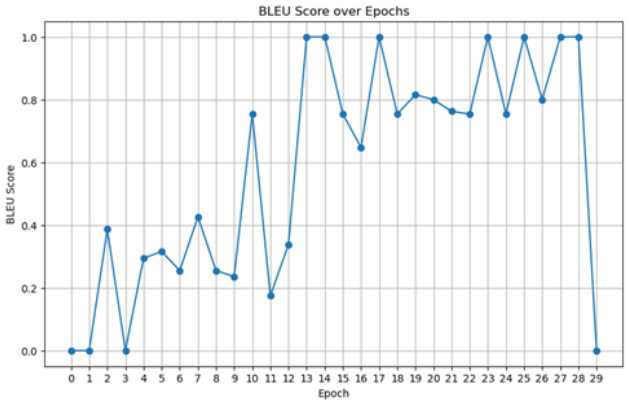
Initialize `bleu_scores` as empty list
for each `epoch` from 0 to 29 **do**
  Load model weights for `epoch`
  Generate `predicted_tokens` using greedy decoding
  Obtain `reference_tokens` from target text
  Compute BLEU score between `predicted_tokens` and `reference_tokens`
  Append BLEU score to `bleu_scores`
Plot `bleu_scores` over epochs



(a) Encoder - Layer 1



(b) Decoder - Layer 1



(c) Encoder - Layer 2

Figure 5.8: BLEU Score Analysis

2. **Refering to figure: 5.8b**

    - If the inference is done multiple times, the final predicted output varies slightly, confirming that LLMs produce a non-deterministic output because it is predicting and not memorizing.

3. **Refering to figure: 5.8c**

- If the model considers a wrong token, it propagates the error in subsequent predictions, resulting in a lower BLEU score.
- The model does not always use the exact words from the target sequence, but may use words with similar meanings, which can still be correct translations of the given source text.
- This shows that BLEU score is not always a reliable metric for evaluating translation quality.

When Does the Model Start Learning?

- As seen from the above graphs, the model builds meaningful relationships with a gradual increase in epochs.
- The learning behavior varies from sentence to sentence and also depends on the order in which sentences are fed to the model.
- This implies that feeding sentences containing similar words as the previously given inputs might help the model learn faster, since it already has an idea of their relationships with other words.
- Generally, relationship-building becomes noticeable between the 10th and 15th epochs during training. Hence, training for 30 epochs allows the model to gather sufficient understanding of each token.
- This method can be further explored to calculate the minimum required epochs to effectively train the model.

Chapter 6

CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

In this study, we successfully elucidated the encoder-decoder architecture and the significance of each component within the transformer model. Using examples, we clarified the fundamental concepts underlying each component, particularly highlighting the attention mechanism. Attention enables the model to store and selectively focus on relevant tokens within sentences, while also facilitating parallel processing during training.

Through detailed visualization techniques, we observed how attention scores fluctuate across different layers and heads, revealing the dynamic relationships between tokens. Graph and cluster analyses demonstrated that certain tokens such as nouns, adjectives, and verbs often act as cluster heads, exerting strong influence over neighboring tokens. In contrast, less critical tokens like pronouns and prepositions shift clusters based on which influential token dominates in a given context.

By applying network analysis methods like the Louvain algorithm and metrics such as Betweenness Centrality, we were able to identify important nodes within token clusters and better understand the structural importance of various tokens. Additionally, evaluation metrics like the BLEU score were used to gauge the quality of the model's output, linking attention behavior to translation accuracy.

6.2 Future Scope

This research opens several avenues for future work:

- Extending interpretability methods to larger and more diverse datasets and transformer variants such as GPT or BERT, to generalize findings across architectures.
- Investigating the impact of pruning or modifying specific attention heads identified as redundant or less important to optimize model efficiency without sacrificing accuracy.

- Developing interactive visualization tools that provide real-time insights into token relationships and attention dynamics during model inference.
- Applying similar graph-based analyses to multilingual models to better understand cross-lingual token interactions and translation behaviors.

BIBLIOGRAPHY

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[3] Sarthak Jain and Byron C. Wallace. Attention is not explanation. *CoRR*, abs/1902.10186, 2019.

[4] Mohaimenul Azam Khan Raiaan, Md. Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874, 2024.

[5] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models, 2024.

[6] Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. A survey of the state of explainable AI for natural language processing. In Kam-Fai Wong, Kevin Knight, and Hua Wu, editors, *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 447–459, Suzhou, China, December 2020. Association for Computational Linguistics.

[7] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey. *ACM Trans. Intell. Syst. Technol.*, 15(2), February 2024.

[8] Chirag Agarwal, Sree Harsha Tanneru, and Himabindu Lakkaraju. Faithfulness vs. plausibility: On the (un)reliability of explanations from large language models, 2024.

[9] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022.

[10] Lakshmi Narayan Pandey, Rahul Vashisht, and Harish G. Ramaswamy. On the interpretability of attention networks, 2023.

[11] Chandan Singh, Jeevana Priya Inala, Michel Galley, Rich Caruana, and Jianfeng Gao. Rethinking interpretability in the era of large language models, 2024.

[12] Erik Cambria, Lorenzo Malandri, Fabio Mercorio, Navid Nobani, and Andrea Seveso. Xai meets llms: A survey of the relation between explainable ai and large language models, 2024.

[13] Parv Modi. English to marathi machine translation dataset. `https://www.kaggle.com/datasets/parvmodi/english-to-marathi-machine-translation-dataset`, 2023. Accessed: 2025-05-17.

[14] S.M. Haider Ali Shuvo. Interpreting decision of self-attention network's in the context of image recognition by efficiently utilizing attention scores. In *2023 6th International Conference on Electrical Information and Communication Technology (EICT)*, pages 1–5, 2023.

[15] Ameen Ali, Thomas Schnake, Oliver Eberle, Grégoire Montavon, Klaus-Robert Müller, and Lior Wolf. Xai for transformers: Better explanations through conservative propagation, Jun 2022.

[16] Samira Abnar and Willem H. Zuidema. Quantifying attention flow in transformers. *CoRR*, abs/2005.00928, 2020.

TURNITIN Report

Purvaja_MTech_Thesis.pdf

ORIGINALITY REPORT

14%	11%	10%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	machinelearningmastery.com Internet Source	1%
2	arxiv.org Internet Source	1%
3	medium.com Internet Source	1%
4	Submitted to Queen Mary and Westfield College Student Paper	1%
5	"ECAI 2020", IOS Press, 2020 Publication	<1%
6	dokumen.pub Internet Source	<1%
7	hdl.handle.net Internet Source	<1%
8	lowyx.com Internet Source	<1%
9	www.geeksforgeeks.org Internet Source	<1%
10	Submitted to Liverpool John Moores University Student Paper	<1%
11	Submitted to Monash University Student Paper	<1%
12	Submitted to mctrajiv Student Paper	<1%

13	pmc.ncbi.nlm.nih.gov Internet Source	<1%
14	Submitted to University of Sydney Student Paper	<1%
15	Submitted to Erasmus University of Rotterdam Student Paper	<1%
16	Mingming Shen, Shaobo Li, Jing Yang, Ansi Zhang, Qiuchen He, Ruiqiang Pu. "A TL_FLAT Model for Chinese Text Datasets of UAV Power Systems: Optimization and Performance", International Journal of Intelligent Systems, 2023 Publication	<1%
17	www.coursehero.com Internet Source	<1%
18	www.interspeech2020.org Internet Source	<1%
19	Submitted to Brunel University Student Paper	<1%
20	d197for5662m48.cloudfront.net Internet Source	<1%
21	ad-publications.informatik.uni-freiburg.de Internet Source	<1%
22	ebin.pub Internet Source	<1%
23	www.irejournals.com Internet Source	<1%
24	Submitted to Fakultet elektrotehnike i računarstva / Faculty of Electrical Engineering and Computing Student Paper	<1%